Universidade de Aveiro
2013

Departamento de Eletrónica,
Telecomunicações e Informática

Ricardo Ferreira Figueira

Módulo para investigação e ensino de sistemas de comunicações digitais

Universidade de Aveiro
2013

Departamento de Eletrónica,
Telecomunicações e Informática

Ricardo Ferreira Figueira

Módulo para investigação e ensino de sistemas de comunicações digitais

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor Paulo Miguel Nepomuceno Pereira Monteiro (orientador), Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Arnaldo Silva Rodrigues de Oliveira (coorientador), Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

presidente	Prof. Dr. José Rodrigues Ferreira da Rocha professor catedrático da Universidade de Aveiro
vogais	Prof.ª Dr.ª Maria do Carmo Raposo de Medeiros professora associada da Universidade de Coimbra
	Prof. Dr. Paulo Miguel Nepomuceno Pereira Monteiro professor associado da Universidade de Aveiro (orientador)

agradecimentos

Em primeiro lugar, agradeço e dedico este trabalho à minha família, em particular aos meus pais, avós e irmã, pelo constante auxílio demonstrado ao longo da minha formação, tanto pessoal como académica, e sem os quais este trabalho não teria sido possível.

Agradeço ao Professor Paulo Monteiro, ao Professor Rui Ribeiro e ao Professor Arnaldo Oliveira pela disponibilidade e orientação à elaboração desta dissertação, nomeadamente nas áreas de sistemas de comunicação e dos sistemas reconfiguráveis.

Deixo também um agradecimento ao Engenheiro Nelson Ribeiro pelas ocasionais trocas de ideias que provaram ser uma mais-valia à realização deste trabalho.

Por fim, agradeço aos colegas pelo companheirismo e apoio demonstrado ao longo destes cinco anos. Em particular, um obrigado ao colega Telmo Cavaco, que me acompanhou desde o início.

palavras-chave

Kit, Sistema Banda-base, PRBS, Formação de Pulso, AWGN, Filtros, Recuperação de informação, FPGA

resumo

Este documento apresenta o projeto de um módulo laboratorial para ensino e investigação de uma vasta gama de sistemas de telecomunicações de banda base, desde os conceitos mais simples até aplicações mais complexas. O módulo foi desenvolvido na Universidade de Aveiro (UA) e no Instituto de Telecomunicações de Aveiro (IT). Os subsistemas digitais que simulam diferentes componentes da transmissão digital num sistema serão apresentados com ênfase na implementação em FPGA (field programmable gate array) de um gerador de sequências binarias pseudoaleatórias, codificação e descodificação de linha, filtros, gerador de ruído gaussiano e circuitos de recuperação de informação.

keywords

Kit, Baseband system, PRBS, Pulse Shaping, AWGN, Filters, Data Recovery, FPGA

abstract

This document presents the design of a laboratorial module for training and research purposes of a wide range of digital baseband telecommunications systems, from the basic concepts through to more complex applications. The module was developed at University of Aveiro (UA) and at Institute of Telecommunications of Aveiro (IT). The digital subsystems that emulate different parts of a digital transmission system will be presented with emphasis on the implementation in a FPGA (field programmable gate array) of the programmable pseudorandom bit sequence (PRBS) generator, line coding and decoding, filters, gaussian white noise generator and data recover circuits.

Content

List of acronyms

AWGN	Additive White Gaussian Noise
CMI	Coded Mark Inversion
DAC	Digital to Analog Convertor
EDK	Embedded Development Kit
FIR	Finite Impulse Response
FIFO	First In First Out
FPGA	Field Programmable Gate Array
IP Core	Intellectual Property Core
ISI	Inter-symbol-interference
LFSR	Linear feedback shift register
LSB	Least Significant Bit
MSB	Most Significant Bit
NRZ	Non-Return-to-Zero
pdf	Probability density function
PLB	Processor Local Bus
PRBS	Pseudo-random bit sequence
PSD	Power spectral density
RZ	Return-to-Zero
SDK	Software Development Kit
SNR	Signal Noise ratio
XPS	Xilinx Platform Studio

List of symbols

M	Fir Filter Order
r	Roll-off Factor
R_s	Symbol Rate
T_s	Symbol Period
F_s	Sampling Frequency
σ	Standard deviation

List of figures

List of tables

1. Introduction

1.1 Motivation

The digital communication systems become more complex in order to improve the transmission quality with higher robustness to the channel impairments and also with higher power and spectral efficiency. The modules that were developed in this project, emulate in hardware the fundamental blocks of a digital communications system, and will provide to students a tool to understand the concepts that are inherent to this kind of system. Several output test probes were made available to the user, enabling the assessment of the communication chain module performance. Depending on the required operating mode, the probe type is selected. A Matlab^{TM} interface between the computer and the hardware will enable a rigorous way of analyze the system data but on the other hand the physical DAC outputs allow the students a real time system analysis.

1.2 Objectives and document structure

As mentioned in the previous section, the main goal of this project is to create a laboratorial kit that will enable a real time analysis of a baseband digital communication system. To achieve it, the designed system has two FPGA's, a Matlab^{TM} communication and 4 channel DAC outputs.

The present document is divided in seven chapters and two appendix sections. It was used an up-to-bottom approach when writing this document, where the level of detail in the system description will increase as we move forward in chapters. In the current chapter, is presented the motivation to this project design and a brief introduction to the chapters that this document comprehends. In second chapter, is presented a high level description of this baseband digital communication system and how the use of an FPGA is relevant for the teaching of this kind of systems. After this quick overview of the system, more detail is added in its description. In the third chapter, each one of the blocks developed in the system (sequence generator, pulse shaping, AWGN generator, filters, data recovery, etc) will be analyzed, without getting into hardware specifications. This approach will let the reader to be familiar with the theoretical concepts, required to design the blocks that are part of the communication chain module. In the fourth chapter, a brief introduction to the hardware and programming tools used in the project is performed, such as the Spartan-6 FPGA and the Xilinx Embedded Development Kit (EDK). Also in this chapter will be described the VHDL algorithms used in the blocks presented in previous chapters.

The designed system comprehends the communication between two FPGAs and Matlab^{TM}. In chapter five are presented the handshake protocols that allow these communications. The description of algorithms will be done using flux and block diagrams, despite the use of the C, Matlab^{TM} and VHDL programming languages in its conception. After the algorithms are implemented in hardware and the required block configuration is done, is possible to test its performance. In chapter six, the different system blocks will be tested and the results will be compared with theoretical results, through Matlab analysis. In the final chapter some conclusions and suggestions of possible future work are presented, given emphasis to possible blocks to be developed within the kit.

In the appendix A are provided two manuals, one for the common user and another for the programmer to interact with the kit. The first one is a higher level manual, which will be based in the interface created in the Matlab^{TM} guide tool. The second one will present the commands needed to initialize the system, without the need of the Matlab, using only a command terminal. Still in this appendix, a small getting started guide is presented. The appendix B is also programmer oriented, since it will describe the filters and FIFO IP Cores configuration, upon its creation in VHDL.

2. System Overview

2.1 Baseband digital communication systems basics

An important goal in the design of a communication system is often to minimize equipment cost, complexity and power consumption while also minimizing the bandwidth occupied by the signal and/or transmission time. Digital communications systems usually represent an increase in complexity over the equivalent analogue systems, however they have become the preferred option for most new systems and, in many instances, have replaced existing analogue systems. This is because, within other reasons, they accomplish for:

- Increased demand for data transmission;
- Increased scale of integration, sophistication and reliability of digital electronics for signal processing, combined with decreased cost;
- Possibility of channel coding (line, and error control) which minimizes the effects of noise, interference and increase the data safety upon transmission.
- Ease with which bandwidth, power and time can be traded off in order to optimize the use of these limited resources [1].

Much of the rest of this dissertation is concerned with the operating principles and performance of a digital baseband communication system formed by a transmitter/receiver pair linked by a communications channel. In this chapter, however, we give only an overview of such a system, incorporating a brief account of what each block in figure 2.1.1 does and why it might be required.

Fig. 2.1.1 depicts the most common blocks present in a digital baseband communication system. In this communication chain, the generated data is already in digital format and it can be from a fixed or a random bit sequence. After the data generation, pulse shaping must occur in order to prepare it to the channel characteristics (for example, bad respond at low frequencies) and to make the receptor synchronism easier. The pulse shaping allow for that reason, a transmission quality control. Besides from elementary pulse shaping, like the raised-cosine spectrum, this block also implements a line coding, such as NRZ, RZ, AMI, bi-phase or Coded Mark Inversion. The type of line coding will define, among other aspects, the efficiency of data transmission and how the clock is recovered at the receiver.

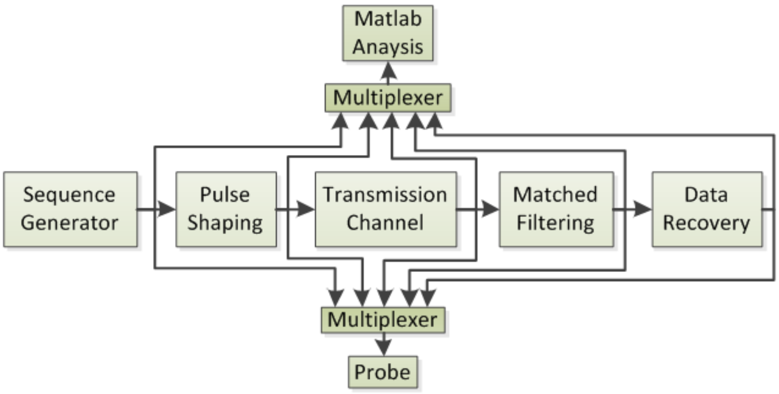


Figure 2.1.1 – Digital Baseband Communication System Blocks

The communications path from transmitter to receiver may use lines or free space. Examples of the former are wire pairs, coaxial cables and optical fibers. Whatever the transmission medium is, it will be at this point that much of the attenuation, distortion, interference and noise are encountered. The nature and severity of transmission medium effects is one of the major influences on the design of transmitters and receivers [1]. After the transmission channel, a low-pass filtering is applied to the signal. This filtering is addressed in Fig. 2.1.1, at the matched filtering block. Also due to transmission channel losses, distortions and the presence of noise, the received signal will be corrupted and therefore a data recovery must be performed. The data recovery block will comprehend the clock recovery, sampling, level decision and line decoding circuits.

The signal at the input and output of different blocks should be available to the user in order to identify and study the entire signal processing along the transmission link. Therefore, the laboratorial module provides along the system several test ports, a Matlab^{TM} communication and a data analyze monitor. The probe, comprehending four channels, will allow the information analysis in real time, through the use of a DAC in several points of the system. The module communicates with an external computer via a standard serial transmission and the software for module management and data acquisition software was implemented in Matlab^{TM}.

Unlike processors, FPGAs use dedicated hardware for processing logic and do not have an operating system. Because the processing paths are parallel, different operations do not have to compete for the same processing resources. This will allow multiple control loops running on a single FPGA device, at different rates. Since this project represent a complex set of blocks and data to be analyzed in real time, an implementation based in FPGA was used, enabling also the testing under real hardware limitations.

3. System Specifications

The system that is presented in this document was designed to allow the user to define some of its parameters. In this chapter, the parameters of each block in the communication chain will be described in a more theoretical approach.

3.1 Sequence Generation

The designed sequence generator is composed by a pseudo-random bit sequence (PRBS) and a programmed sequence generator which can be selected according to the type of data that is to be simulated in the communication chain.

3.1.1 Specifications

As it can be seen in figure 3.1.1, the inputs of the sequence generator block are the number of cells (*#Cells*), seed, clock frequency, mode and number of words (*#Words*).

After the PRBS and the programmed sequence configuration, the mode of operation must be selected. The values that are generated by either the PRBS or by the programmed sequence must go through a serial communication before reaching the Matlab^{TM}. These intermediate steps will allow a lower bit rate than the rate permitted by the generators themselves. Therefore, two modes of operation were created: *Continuous* - the clock frequency is chosen by the user and the limit is the maximum rate which the generator can produce the values to be sent to the DAC; *Step-by-Step* - the values generated through this second mode are stored temporarily inside the FPGA and sent to Matlab^{TM} at the maximum rate allowed by the UART and processor operation. The input parameter, *#Words*, is defined by the user as the number of samples that will be sent to the Matlab^{TM} for analysis, and is only relevant in the *Step-by-Step* mode. Is important to refer that the maximum value that the *#Words* parameter may take, will be equal to the size of the storage element inside the FPGA.

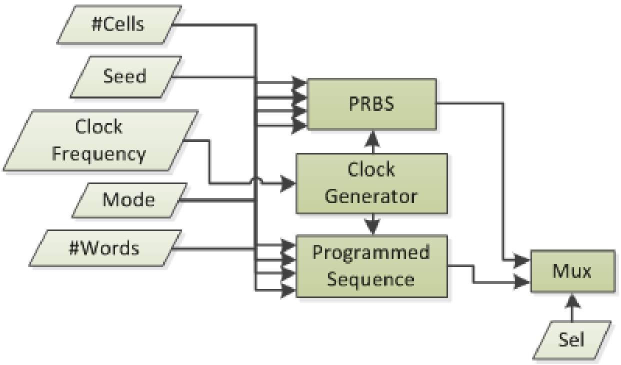


Figure 3.1.1 – Sequence Generator Internal Blocks

The last step is the selection (*sel* input) between PRBS generator and the programmed data sequence, to send their output values outside the generator.

3.1.2 PRBS

A linear feedback shift register (LFSR) is a key component in PRBS generation. As so, in this section it will be discussed the operational principles of this circuit. At the end, it will be presented some tables with the feedback taps to get the maximum length sequences (m sequences).

3.1.2.1 LFSR implementation

The LSFR generator, as the name suggests, is composed of a shift register in which the content of its cells must be updated according with a certain rule. This rule must generate a linear combination of the cells in the current state and then shift the result to one end of the current state vector [2].The linear operation comprises the arithmetic modulo-2 (logical XOR operation).

There are two methods to implement LSFR. In first one, the result of the cells linear combination will be placed in one end of the register and the remaining cells are just shifted (fig. 3.1.2a). Translating to equations we have in this case:

$$\begin{aligned} a'_0 &= \left(\sum_{j=1}^{n-1} c_j a_j \right) + a_n \\ a'_i &= a_{i-1}, \qquad i = 2, \ldots, n \end{aligned} \qquad (3.1.1)\ [2]$$

, where *a* and *a'* represent the current and the new state vector of the register, respectively. For a left-shift operation (as depicted in fig. 3.1.2a), a0' represents content the right-most bit and for a right shift operation is the left most bit. One of the necessary conditions for a full length sequence be generated by a LFSR is that $c_n = 1$. In other words, the left most bit must always be part in the feedback result.

In the second method, the selected bits will be changed according to the XOR operation between the left-most bit and the bit at his right – fig. 3.1.2b

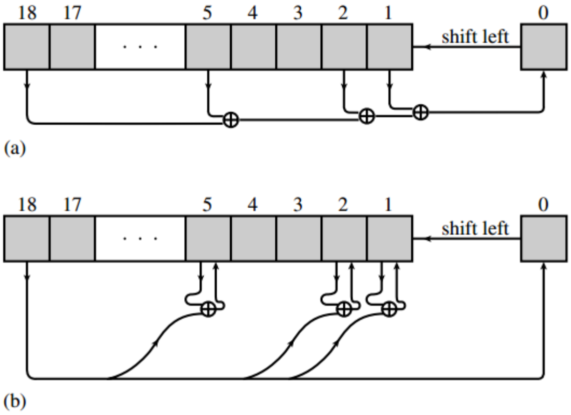


Figure 3.1.2 – Two methods for LSFR implementation [2]

Usually, the first method is more efficient to be implemented in common hardware, but on a FPGA, will force to a cascade of *n* XOR logic operations (where n is the number of cells in the shift register) which will lead to a greater propagation delay than the later one. Therefore, the second method was chosen for the FPGA implementation.

The mathematical proprieties of equation (3.1.1) derive from the proprieties of a polynomial equation:

$$P(x) = x^n + c_{n-1}x^{n-1} + \cdots + c_2x^2 + c_1x + 1 \qquad (3.1.2)\ [2]$$

, where each c_j can take the values 0 or 1 (jth cell that takes part on the feedback loop). In equation (3.1.2), the notation of the c_j's started in the index number 1 because the c_0 (like c_n) is always equal to 1.

3.1.2.2 Taps Notation

An alternative way to express the polynomial presented in equation (3.1.2) is the representation of only the nonzero x coefficients (often referred as LFSR's taps). It can also be abbreviated for the powers of the polynomial. Considering the polynomial $x^{18}+x^5+x^2+x+1$, we have the powers given by: (18, 5, 1, 0).

The maximum length sequence (*m* sequence) that a LFSR can generate is equal to the number of non-zero different states $m=2^n-1$, where *n* is the number of cells in the shift register. The only state that is not comprised is the all-zero's state, since the state machine will be at this state forever. To achieve an *m* sequence, an irreducible and primitive polynomial must be selected. In the work that is being developed, the data generator can be set for a PRBS with length from 2^3-1 up to $2^{32}-1$. A table with the primitive polynomials to generate these PRBS is provided below:

n (register size)	Left – shift	Right - Shift
3	(3,1,0)	(3,2,0)
4	(4,1,0)	(4,3,0)
5	(5,2,0)	(5,3,0)
6	(6,1,0)	(6,5,0)
7	(7,1,0)	(7,6,0)
8	(8,4,3,2,0)	(8,6,5,4,0)
9	(9,4,0)	(9,5,0)
10	(10,3,0)	(10,7,0)
11	(11,2,0)	(11,9,0)
12	(12,6,4,1,0)	(12,11,8,6,0)
13	(13,4,3,1,0)	(13,12,10,9,0)
14	(14,5,3,1,0)	(14,9,11,13,0)
15	(15,1,0)	(15,14,0)
16	(16,5,3,2,0)	(16,14,13,11,0)
17	(17,3,0)	(17,14,0)
18	(18,5,2,1,0)	(18,17,16,13,0)
19	(19,5,2,1,0)	(19,18,17,14,0)
20	(20,3,0)	(20,17,0)
21	(21,2,0)	(21,19,0)
22	(22,1,0)	(22,21,0)
23	(23,5,0)	(23,18,0)
24	(24,4,3,1,0)	(24,23,21,20,0)
25	(25,3,0)	(25,22,0)
26	(26,6,2,1,0)	(26,25,24,20,0)
27	(27,5,2,1,0)	(27,26,25,22,0)
28	(28,3,0)	(28,25,0)
29	(29,2,0)	(29,27,0)
30	(30,6,4,1,0)	(30,29,26,24,0)
31	(31,3,0)	(31,28,0)
32	(32, 7, 5, 3, 2, 1, 0)	(32, 31, 30, 29, 17, 15, 0)

Table 3.1.1 - Primitive polynomials for left-shift LSFR and right-shift LSFR [2]

However, the polynomials presented in [2] are only for left-shift case, considering the less significant bit at the right. To determine the powers that will lead to a maximum length sequence, for the right shift case, we subtract to *n* each one of left-shift powers. Considering the example mentioned above, we have: Left-shift case: (18,5,2,1,0); Right-shift case: (18-18,18-5,18-2,18-1,18-0)=(0,13,16,17,18)≡(18,17,16,13,0). The powers for right-shift are shown in table 3.1.1 and will be those used in the PRBS implementation.

The determination of the primitive polynomials is done by factoring the polynomial $x^{2^n-1}+1$ and finding which of the resulting polynomials are primitive (have order equal to *n*). Since this requires a modulo-2 linear algebra (probably with the aid of a computer algorithm), the tables presented above are frequently obtained from a book (as is the case in this text).

After the connections have been made, the circuit is prepared to start the sequence. An initial state must then be loaded into the register to start the operation. This state, often called as *seed*, has the same size of the register and must not take the all-zeros combination.

3.1.3 Programmed Sequence

Other easy way to generate a sequence is to start with a known *seed* and just shift it to one direction, which will lead to a pattern that will repeat itself through a circular shift. This is how the programmed sequence works and if noticed, is the PRBS without the feedback taps in the shift register. Both the PRBS and the programmed sequence will then generate samples that meet the user specifications.

3.2 Pulse Shaping

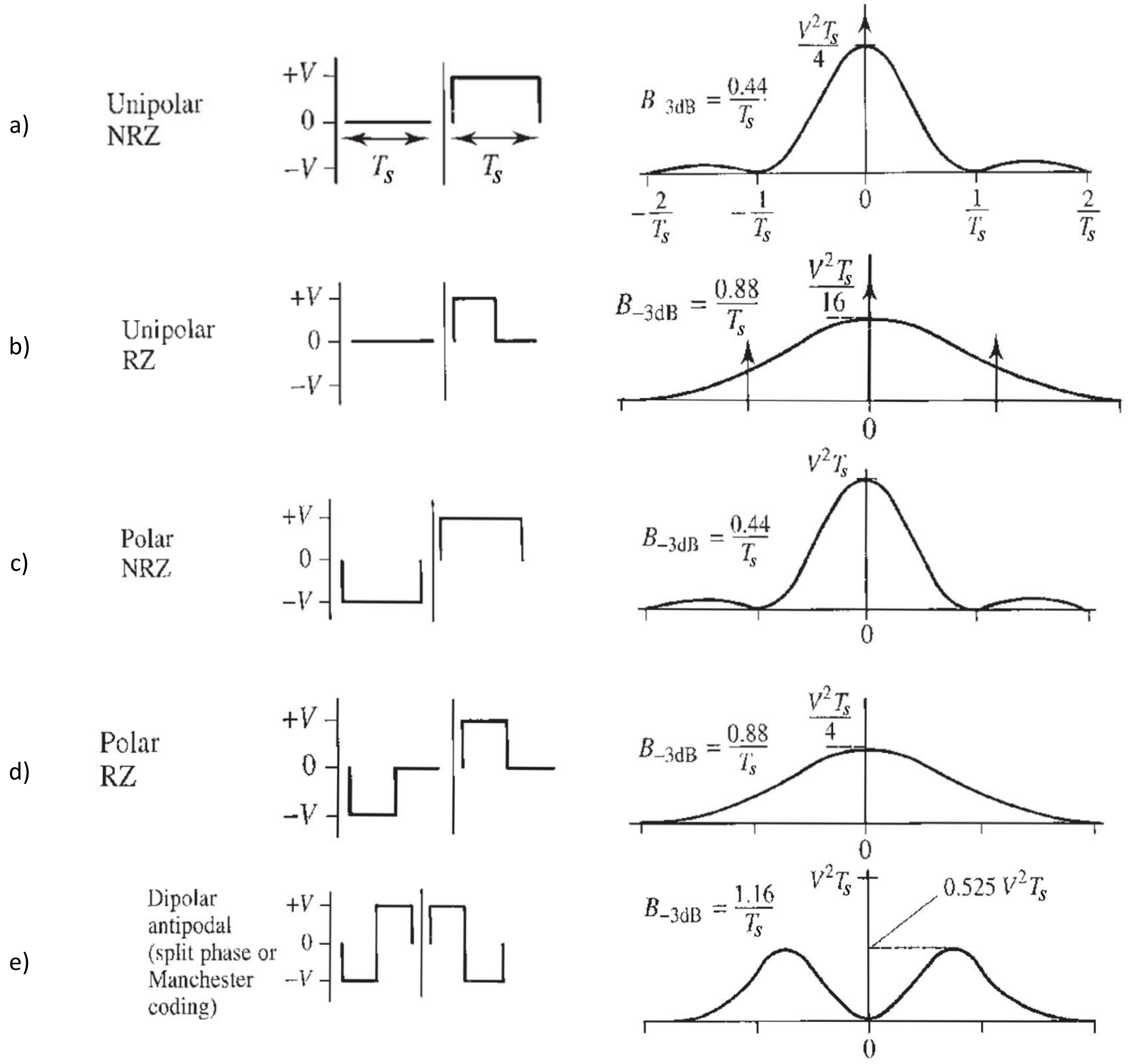
3.2.1 Line Coding

Many of the baseband digital signals have a high content at low frequencies. These signals would be highly affected if they were sent directly through transmission channels with bad response to low frequencies. The spectrum of the transmitted signal in a digital system depends of the elementary impulse shape and of the statistical proprieties in all the impulses sent in the sequence. The line coding will generate a new signal which will reshape each pulse. Besides the spectrum adaptation to the transmission channel frequency response (power spectral density), there are other proprieties that the line coding might have:

- Presence or absence of a DC level;
- Ease of clock signal recovery for symbol synchronization;
- Possibility of error detection and correction;
- Efficiency – the signal bandwidth and transmitted power should be as low as possible;
- Transparency –the property that any arbitrary symbol, or bit, pattern can be transmitted and received correctly, independently of the 1's and 0's original bit sequence;
- BER performance – relative immunity to noise.

In Fig. 3.2.1 is presented a comparison between several line codes that were implemented in this project: Unipolar NRZ/RZ, Polar NRZ/RZ, Manchester, Bipolar NRZ/RZ and CMI (coded-mark-inversion) at time and frequency domain.

Pulse shape in time domain **Pulse spectral content**



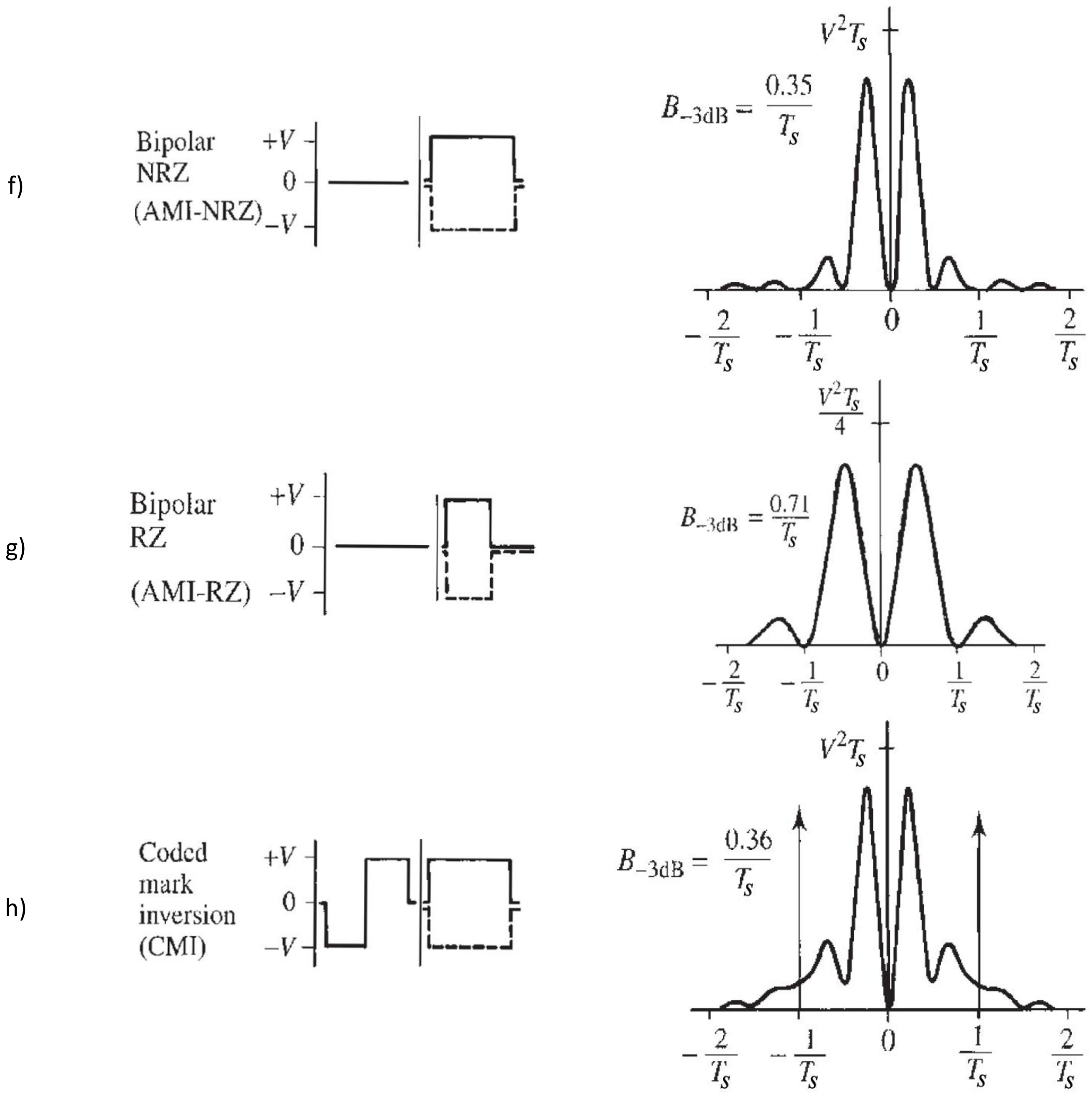


Figure 3.2.1 - ‘0’ and ‘1’ symbols in time domain and spectrum representation [1]

In unipolar signaling (fig. 3.2.1a and b), also called OOK (on-off keying), the binary symbols ‘0’ and ‘1’are represented respectively by the absence and presence of a pulse (with voltage level +V), respectively. If the duration of the pulse is equal to one symbol period, T_s, the signal is called NRZ (non-return to zero). On the other hand if the pulse duration is equal to a fraction of the symbol period, the signal is called RZ (return to zero).

The difference between the polar signaling and the unipolar case is that, in the first case, the symbol ‘0’ is also represented by one pulse, but with an inverse voltage level (represented as ‘–V’ in Fig. 3.2.1).

The dipolar signaling (Fig. 3.2.1e) is characterized by their spectral null at 0Hz and vanishes in its vicinity, propriety that is not found in the unipolar and polar signaling, which is a result of the total area under the pulses of symbols ‘0’ and ‘1’ be equal to 0. This aspect enables the signal to be sent through an AC coupled transmission line. However, this type of line coding does not have a spectral line at clock frequency (1/T_0 Hz) as we can find in an OOK-unipolar RZ signaling.

The bipolar signaling (Fig.3.2.1f and g), also called AMI (Alternate-Marked-Inversion), like the polar and dipolar code, uses three voltage levels (-V, 0, +V). The ‘0’ symbol is respectively translated into 0 voltage level (as the unipolar case) and the ‘1’ symbol as ‘+V’ or ‘–V’ voltage (if the previous ‘1’ bit was sent as +V, the next bit will be sent as –V, and vice-versa). This type of coding enables not only the presence of a null at 0Hz (and fade out at its vicinity), but also a decrease in bandwidth over the one required for the unipolar and polar case.

The last code that will be analyzed is the CMI. His shape is a mix between the AMI-NRZ (for ‘1’ symbols) and the dipolar signaling (for ‘0’ symbols). Is possible to see in Fig.3.2.1h, that the CMI manage to have a spectral line at clock frequency (1/T_0 Hz), at the same time that it requires a small bandwidth and extinguishes at low frequencies (as in the previous dipolar and bipolar cases).

3.2.2 Raised Cosine Filter

Through a real digital system, the ideal rectangular impulses considered so far, have a spectral content that extends to infinite frequencies. Since real channels have a limited bandwidth, a portion of the signal spectrum will be suppressed, which cause the pulse to spread in time domain. This spread phenomenon will lead to inter-symbol-interference (ISI) which may result in errors at the sampling and decision process, forward in the communication chain. The obvious solution is to send band-limited pulses, so that they can be transmitted over a band-limited channel. However, band-limited pulses cannot be time-limited. One example of this type of pulses is an ideal low pass filter in frequency domain, which results in a *sinc* pulse in time domain (fig. 3.2.2a). The *sinc* pulse is zero in all sampling instants except in one, being for that reason an ISI-free signal. It enables the transmission of R_s = 2B symbols/second using a channel with bandwidth B, with ISI = 0. This R_s rhythm is the maximum transmission rate with ISI=0, and is named the Nyquist rate. Figure 3.2.2b shows, through an example, how null ISI is achieved with the *sinc* pulse (considering once more T_s as the symbol period).

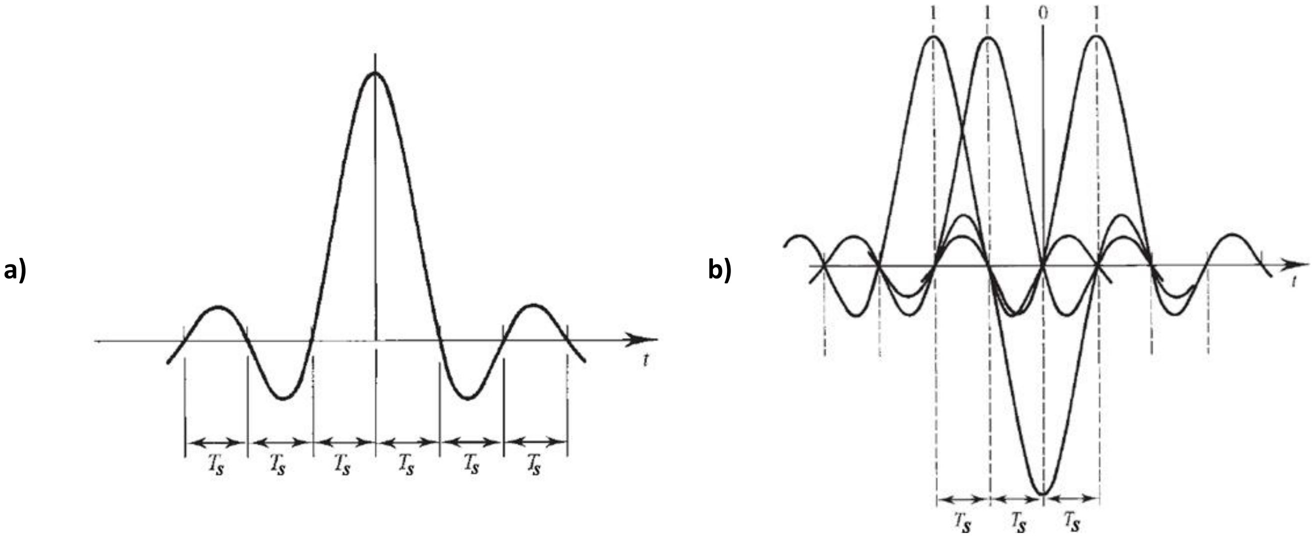


Figure 3.2.2 – a) Sinc pulse in time domain and b) sinc pulse signaling for the symbol sequence: 1,1,0,1 Adapted [1]

However, this pulse is impractical because it starts at -∞, and one must wait an infinite time to generate it. Any attempt to truncate it would increase its bandwidth beyond B = $R_s/2$ [3]. Even if this pulse were realizable, any deviation of R_s, sampling rate or the presence of jitter could lead to a major ISI due to the slow pulse decay (∝ 1/t). This scheme therefore fails unless everything is perfect, which is a practical impossibility [3].

In practice, due to those difficulties, are used other pulse shapes. In his first criterion, Nyquist achieve null ISI, by choosing a pulse which has a nonzero at its center (t=0) and zero amplitude at $t = \pm nT_s$, where T_s is the separation between successive transmitted pulses:

$$p(t) = \begin{cases} 1, \quad t = 0 \\ 0, \quad t = \pm nT_s \left(T_s = \dfrac{1}{R_s}\right) \end{cases} \qquad \text{(3.2.1) Adapted [3]}$$

In order to have a low ISI in non-ideal sampling, the pulse must also have a faster decay than the *Sinc* pulse but satisfies at the same time the propriety presented in (3.2.1). Nyquist has shown that such a pulse requires a bandwidth of $k.\frac{R_s}{2}$, with $1 \leq k \leq 2$. A pulse that satisfies these conditions, known as the raised cosine pulse, is presented in equations (3.2.2) and (3.2.3) and in figures 3.2.4 and 3.2.5.

$$P(w) = \begin{cases} \frac{1}{2}.\left\{1 - \sin\left(\frac{\pi\left[w - \left(\frac{w_s}{2}\right)\right]}{2.w_x}\right)\right\} \\ 0, & |w| > \frac{w_s}{2} + w_x \\ 1, & |w| < \frac{w_s}{2} - w_x \end{cases}, \quad \left|w - \frac{w_s}{2}\right| < w_x$$

(3.2.2) Adapted [3]

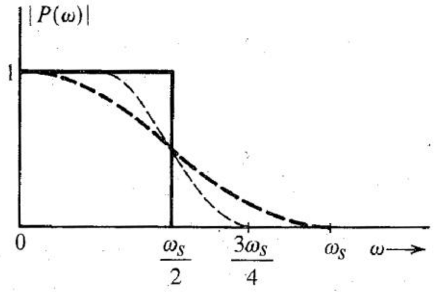


Figure 3.2.4 - Frequency domain pulse Adapted [3]

$$p(t) = R_s.\left(\frac{\cos(2\pi.f_x.t)}{1 - (4.f_x.t)^2}\right).sinc(\pi.R_s.t)$$

(3.2.3) Adapted [3]

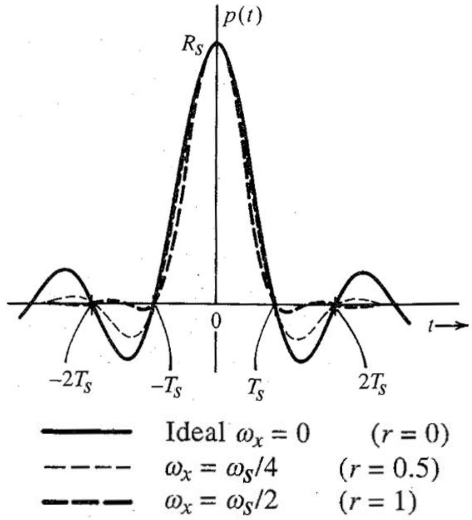


Figure 3.2.5 - Time domain pulse Adapted [3]

Where $w_s = 2\pi R_s = \frac{2\pi}{T_s}$ (rad/s) and $r = \frac{2w_x}{w_s}$ (ratio between the excess bandwidth, w_x, and the theoretical minimum bandwidth, $\frac{w_s}{2}$). Therefore, the bandwidth of *P(w)* is : $B_T = \frac{R_s}{2} + f_x = \frac{R_s}{2} + \frac{r.R_s}{2} = \frac{(1+r).R_s}{2}$ and the constant r is called the roll-off factor, that is $0 \leq r \leq 1$ or in percent: $0\% \leq r \leq 100\%$. For example, if *P(w)* has a bandwidth 50% higher than the theoretical minimum, then $r = 0.5 = 50\%$. As a result, $0 \leq w_x \leq \frac{w_s}{2}$.

One can find several characteristics of the raised cosine, by analyzing fig. 3.2.4 and 3.2.5. The first obvious characteristics are:

- A maximum bandwidth of R_s;
- For r=1, the pulse in the time domain has zero values not only at multiples of the symbol period, T_s, but also at points midway between these instants;
- A fast decay proportional to $\frac{1}{t^3}$. As large as the roll-off factor is, the fastest the pulse will decay, but bigger the bandwidth will be. Therefore, a trade-off must be made, given the channel characteristics.

The fast time decay and the smooth frequency transitions make the raised cosine pulse realizable in practice.

3.2.3 Fir Filter

In order to implement the raised cosine filter (mentioned in the section 3.2.2) as well as several other filters in the channel and in the receiver, a FIR (Finite Impulse Response) filter was used in this project.

As seen in equation (3.2.4), each output sample, y[n], of the system is given by the sum, L, of a finite number of weighted samples, M+1, of the input sequence, $x[n-1]$, where n is the sample number and b_k is the k^{th} filter's coefficient.

$$y[n] = \sum_{k=0}^{M} b_k \,.\, x[n-k]$$

(3.2.4) [4]

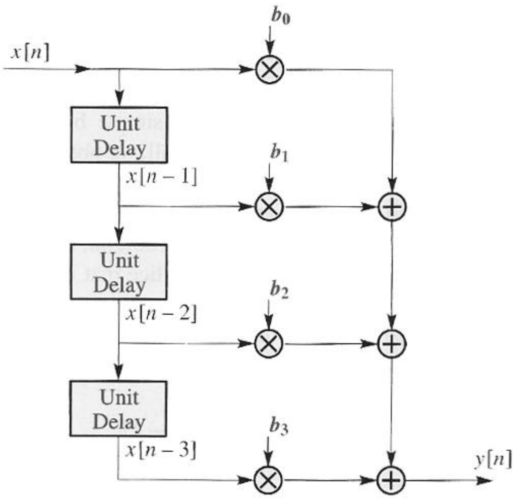


Figure 3.2.6 – Block diagram structure for a third-order fir filter [4]

Since (3.2.4) does not involve future values of the input, the system is causal and therefore the output cannot start before the input is nonzero. In the filter implementation, one must be aware of this limitation and wait for the generation of M+1 input samples, so that a correct output value at the present sample is obtained. A similar effect will happen when the input samples stop being generated, due to their finite sequence proprieties.

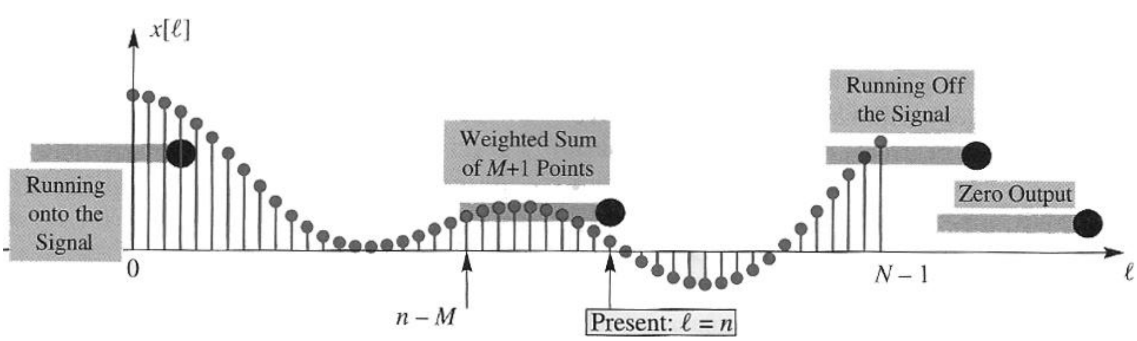


Figure 3.2.7 –Operation of an M^{th} order causal FIR filter showing various positions of the sliding window of M+1 points under which the weighted average is calculated [4]

In figure 3.2.7 is depicted the condition mentioned before, where the filter's sliding window position will determine which input samples will be used in the weighted average.

When the system input is the unit impulse sequence (eq. 3.2.5), the filter output is named as unit impulse response (or just impulse response) and denoted as y[n] = h[n]. In this case, the output will be non-zero only when k=n. In other words, h[n] is just the filter coefficient sequence, b_k. Therefore, the impulse response will characterize the filter to be designed.

$$x[n] = \delta[n] = \begin{cases} 1, n = 0 \\ 0, n \neq 0 \end{cases} \tag{3.2.5}$$

It is common practice to determine the impulse response of the filter from the theoretically desired impulse response. When necessary, the target impulse response is time truncated, time shifted and sampled, in order to become time finite, causal and discrete. In this project, all the filter coefficients, such as the raised cosine, were generated with the help of Matlab. The number of coefficients of the filter will determine the number of samples to be taken from the raised cosine time domain shape. Next, those coefficients are sent to the FPGA which will send them to the proper block in the communication chain.

3.3 Transmission channel

One of the key points in the project is the transmission channel simulation. This will be done through the use of simple filters such as high-pass, and more complex ones that will try to resemble the real conditions, such as the coaxial cable. Furthermore, an Additive White Gaussian Noise (AWGN) was integrated in this system to cope with channel and receiver noise.

3.3.1 Coaxial cable

A coaxial cable is a commonly-used type of transmission line used in the systems of interest and gives raise to results which are very interesting from the pedagogical point of view. These cables exhibit frequency-dependent loss and delay due to distributed parameters in the cable. An infinitesimal length of electrical transmission line can be modeled as a resistance (R in Ω/m) and inductance (L in H/m) in series, and a capacitance (C in F/m) and conductance (G in S/m) in parallel [5], as shown in figure 1 below:

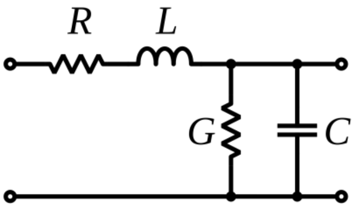


Figure 3.3.1 – Infinitesimal length of electrical transmission line

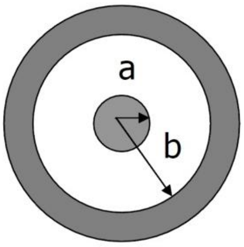


Figure 3.3.2 – Transversal view of a coaxial cable

The shunt capacitance per unit length is independent of frequency and is given by

$$C = \frac{2\pi\epsilon}{\ln\left(\frac{b}{a}\right)}, \qquad \text{(3.3.1) [5]}$$

where ϵ is the permittivity of the medium between the inner and outer conductor, a and b are the radii of the inner and outer conductor, as shown in figure 3.3.2. The series inductance per unit length accounts for two sources of inductance and is given as:

$$L = L_0 + L_{s0}, \qquad \text{(3.3.2) [5]}$$

where L_0 (3.3.3) is the ideal inductance associated with the magnetic component of the field between the conductors and L_{s0} (3.3.4) is the frequency-dependent inductance associated with the magnetic component of the field interior to the inner and outer conductors, due to the imperfect conductivity [5].

$$L_0 = \frac{\mu}{2\pi}\ln\left(\frac{b}{a}\right) \qquad \text{(3.3.3) [5]}$$

$$L_{s0} = \frac{\mu^{1/2}}{4\pi^{3/2}}\left(\frac{\sigma_a^{-1/2}}{a} + \frac{\sigma_b^{-1/2}}{b}\right).f^{-1/2}, \qquad \text{(3.3.4) [5]}$$

where μ is the permeability of the medium between the inner and outer conductor. The series resistance per unit length arises from the same current associated with Ls0. For good conductors, the real and imaginary parts of the wave impedance are equal [5], therefore:

$$R = 2\pi.L_{s0}.f \qquad \text{(3.3.5) [5]}$$

If properly terminated at both ends of the transmission line, the transfer function from the input voltage (i.e., the voltage at the beginning of the transmission line) to the output voltage (i.e., the voltage at distance l) is

$$H(w) = e^{-\gamma l}, \tag{3.3.6) [5}$$

where γ is the propagation constant and is given by:

$$\gamma = \sqrt{(R + jwL).(G + jwC)} \tag{3.3.7) [5}$$

Typically, the shunt conductance is negligible for well-designed transmission line. Thus the propagation constant can be written as

$$\gamma = \sqrt{(R + jwL).(jwC)} \tag{3.3.8) [5}$$

Knowing the transfer function, $H(w)$, is possible to get the impulse response, $h(t)$, through the inverse Fourier transform. As the other filters in the communication chain, the coaxial cable impulse response taps will then be sent to the FIR filter.

3.3.2 AWGN

As referred before, the AWGN will be part of the transmission channel, next to the channel filter. In this section, the designed AWGN will be described using the transformation method presented in [2]. According to this method, is possible to generate a random deviate, y, from a know probability distribution p(y) = f(y), for some positive function f whose integral is 1. The indefinite integral, F(y), of p(y) must be known and invertible. The figure 3.3.3 below depicts the transformation method: a uniform random variable x is chosen between 0 and 1. Its corresponding y on the definite-integral curve is the desired deviate.

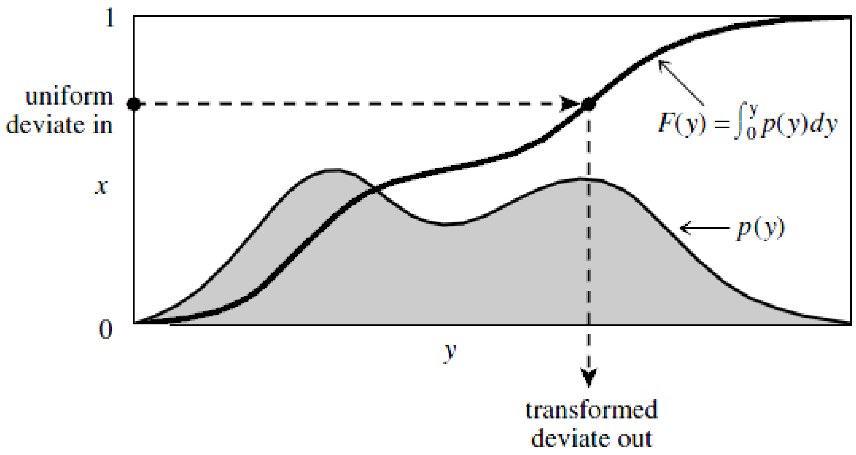


Figure 3.3.3 – Transformation method [2]

After F(y) is obtained, an inversion of this function must occur, F^{-1}, in order to obtain the y values:

$$y(x) = F^{-1}(x) \tag{3.3.9}$$

In the present case, f(y) is a normalized Gaussian distribution: N (0, 1):

$$f(y) = \frac{1}{\sqrt{(2\pi)}} . \mathrm{e}^{\frac{-\mathrm{y}^2}{2}} \tag{3.3.10}$$

As it will be seen forward in chapter 4, a ROM will be used to contain the pre-computed values of function F^{-1}. Since f(y) is an even function, is possible to use only the positive side f(y) to the calculations, decreasing at the same time the number of ROM positions. However, one must also ensure that his integral is equal to 1. This condition can be satisfied simply by multiplying f(y) by two and doing a variable substitution $z = |y|$, obtaining:

$$f(z) = \begin{cases} 0\,, & z \leq 0 \\ \sqrt{\frac{2}{\pi}} . \mathrm{e}^{\frac{-\mathrm{z}^2}{2}}, & z > 0 \end{cases} \tag{3.3.11}$$

$$erf(x) = \frac{2}{\sqrt{\pi}} . \int_0^{\mathrm{x}} \mathrm{e}^{-\mathrm{t}^2} dt \tag{3.3.12}$$

To obtain F(z), the Gauss error function is used (3.3.12), obtaining:

$$F(z) = \int_{-\infty}^{\mathrm{z}} f(z)\, dy = \int_0^{\mathrm{z}} \sqrt{\frac{2}{\pi}} . \mathrm{e}^{\frac{-\mathrm{z}^2}{2}} dz = \frac{2}{\sqrt{\pi}} . \int_0^{\frac{\mathrm{z}}{\sqrt{2}}} . \mathrm{e}^{-\mathrm{t}^2} dt = \mathrm{erf}\left(\frac{z}{\sqrt{2}}\right) \tag{3.3.13}$$

$$F(z) = \begin{cases} 0\,, & z \leq 0 \\ \mathrm{erf}\left(\frac{z}{\sqrt{2}}\right), & z > 0 \end{cases} \tag{3.3.14}$$

The inverse function is then given by:

$$x = F(z) = \mathrm{erf}\left(\frac{z}{\sqrt{2}}\right) \Rightarrow z = F(x)^{-1} \Longrightarrow z = erfinv(x). \sqrt{2} \tag{3.3.15}$$

A non-uniform quantization of the segment x = [0, 1] must be done, through a recursive partitioning of the segment [0, 1]. As mentioned, to reduce the complexity in hardware, a quantized version of (3.3.15) using pre-computed values will be loaded into several ROM's. The number, K, of ROM's will depend on the amount of times the partition will be done in the [0, 1] segment.

3.3.3 SNR

In order to control the signal to noise ratio in the system (SNR), several parameters must be determined: the noise *PSD* (power spectral density, η), receptor's filter noise equivalent bandwidth (B), average signal power (P_s) and sampling frequency (F_s).

The average signal power will depend of data transmitted line coding. Its value over a period is determined by expression number 3.3.16:

$$P_s = \frac{1}{T}\int_0^T s^2(t)\,dt = P_0.A_0^2 + P_1.A_1^2, \tag{3.3.16}$$

, where s(t) is the data signal over time, A_1 and A_0 are the signal amplitude and P_1 and P_0 are the probabilities for the 1 and 0 bits, respectively. However, equation (3.3.16) is only valid if the amplitude is constant over the bit period. For example, the polar NRZ line coding have: $A_1 = A$ and $A_0 = -A$. For $P_1 = P_0 = \frac{1}{2}$, we have: $P_s = \frac{1}{2}.A^2 + \frac{1}{2}.(-A)^2 = A^2$.

In Table 3.3.1 is presented the average signal power for the several line codes used in this project, where A is the absolute maximum amplitude of the data signal:

Line Coding	P_s
Unipolar NRZ	$A^2/2$
Unipolar RZ	$A^2/4$
Polar NRZ	A^2
Polar RZ	$A^2/2$
Manchester	A^2
Bipolar NRZ	$A^2/2$
Bipolar RZ	$A^2/4$
CMI	$A^2/2$

Table 3.3.1 – Average signal power for different line coding formats

In figure 3.3.4 are presented the communication chain blocks used in the system.

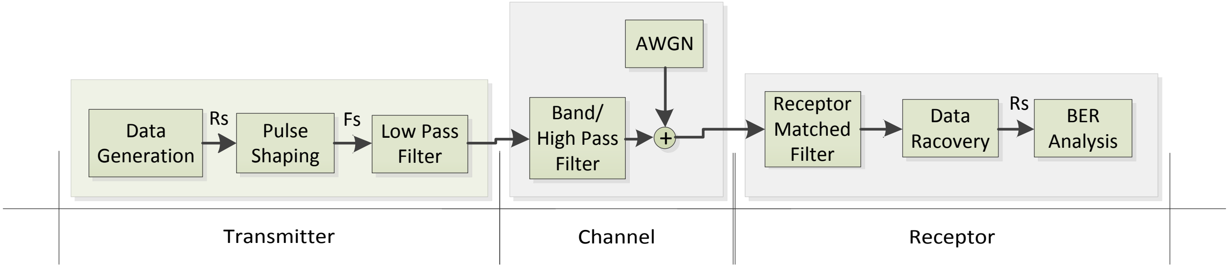


Figure 3.3.4 – Communication Chain Blocks

The noise spectral content will depend on its place in the chain, as shown in figure 3.3.5.

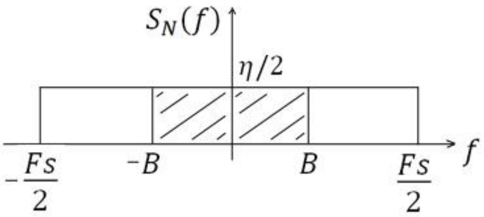


Figure 3.3.5 – Noise spectral content, where $\eta/2$ is the bilateral power spectral density

Considering the noise power in two different points: before, given by (3.3.18), and after, given by (3.3.17), the receiver filter:

$$\begin{cases} (3.3.17)\ P_N = {\sigma^2}_R = \eta.B = \dfrac{P_S}{SNR} \\ (3.3.18)\ P_N = \sigma^2 = \eta.\dfrac{Fs}{2} \end{cases} \Rightarrow \begin{cases} \eta.B = \dfrac{P_S}{SNR} \\ \eta = \dfrac{\sigma^2.2}{Fs} \end{cases}$$

The SNR after the receiver filter will then be given by the ratio:

$$SNR = \frac{P_S.Fs}{B.\sigma^2.2} \tag{3.3.19}$$

The user will set the parameters: Fs, B and η. The receiver bandwidth must be set between the range $[\frac{R_s}{2}, 4R_s]$, where R_s is the symbol rate. On the other hand, F_s and η must verify the noise standard deviation range: $\sigma = \sqrt{\eta.\frac{F_S}{2}}\ \epsilon\ [\frac{A}{8}, 3A]$. The noise cannot be higher than three times the data signal amplitude (A), which gives a maximum limit of 4A for the signal sent over the chain. F_s will be a parameter for advanced users and $\frac{\eta}{2}$ (bilateral spectral density) will be set by the user and have W/Hz as units.

3.4 Data Recovery

After the data has gone through the receiver filter (point A in fig. 3.4.1) the original bit stream must be recovered. To achieve this, several steps must be performed, such as the clock recover, sampling, level decision and line decoding. In this section, a brief description will be done of each one of these blocks, depicted in figure 3.4.1 below.

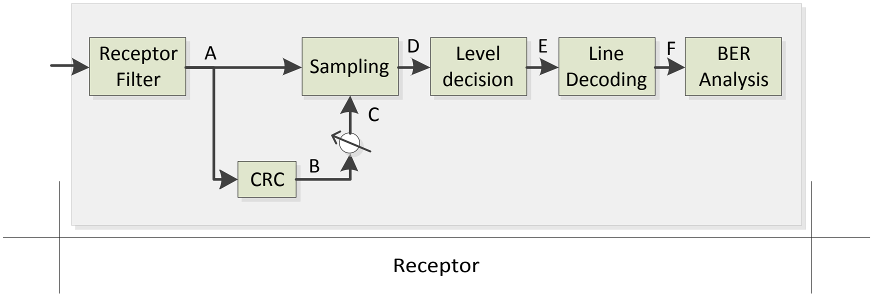


Figure 3.4.1 – Receiver Block Diagram

3.4.1 Clock Recover and Sampling

The clock recovery circuit, CRC, will be responsible to identify the first instants from which the data signal can be sampled later on in the sampling block. In order to do it, it must first detect the data signal edges, this is, the instants where the data signal changes it value. In order to do it, first a level decision is applied to the data signal in point A (it that can take 4095 different values). This decision output is then transformed into a three or two level signal (depending on the line coding). The signal obtained in point B is then used as an initial mark to choose the sampling instants though a phase setting by the user. There must be at least two sampling instants for each symbol, so that it will be possible to identify all the transitions in a return-to-zero signal.

More details about these two blocks are described in the hardware implementation section 4.10, where it will be possible to understand better the interaction between all the signals involved.

3.4.2 Level Decision

After the signal being sampled at the proper instants, a level decision (identical to the one done in the CRC block) must be performed. The user has to set two values in this block: the upper and the lower threshold. If the data signal (point D) is greater than the upper threshold, the resulting signal (point E) will correspond to the “V+” level. If the point D signal is lower than the lower decision threshold, the signal in point E will take the “V-“ level. If it is neither of these cases, the signal will take the “0” level. However, if the data signal has only two levels, the lower decision threshold will not take effect and any sample lower than the upper decision threshold will be set to the “0” level.

Bayes’s decision criterion, described in [1], represents a general solution to setting the optimum reference or threshold voltage, v_{th}, in a receiver decision circuit. The threshold voltage which minimises the expected conditional “cost” of each decision is the value of v which satisfies (3.4.1).

$$\frac{p(v|0)}{p(v|1)} = \frac{C_0.P(1)}{C_1.P(0)} = L_{th} \tag{3.4.1}$$

, where:

- C_0 - lost of information “cost” when a transmitted digital 1 is received in error as a digital 0;
- C_1 - lost of information “cost” when a transmitted digital 0 is received in error as a digital 1;
- $P(0)$ - a priori probability of transmitting the digit 0;
- $P(1)$ - a priori probability of transmitting the digit 1;
- $p(v|0)$ – conditional probability density function of detecting voltage v given that a digital 0 was transmitted;
- $p(v|1)$ - conditional probability density function of detecting voltage v given that a digital 1 was transmitted.
- L_{th} - likelihood threshold

If $L_{th} = 1$, such as would be the case for statistically independent, equiprobable symbols ($P(0) = P(1)$) with equal error costs ($C_0 = C_1$), then the voltage threshold would occur at the intersection of the two conditional pdf’s [1]. This case is known as the maximum likelihood decision criterion, and is represented in figure 3.4.2.

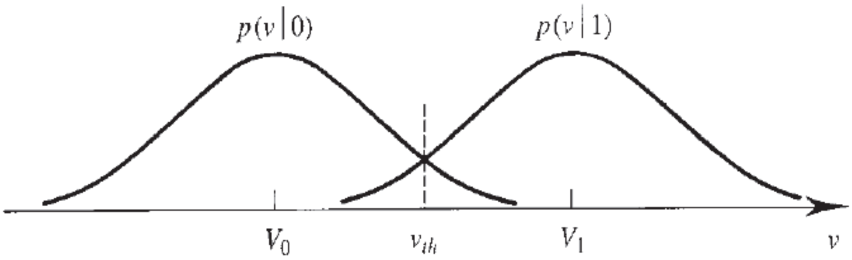


Figure 3.4.2 - Probability distributions for binary transmissions, where V_0 or $V_1$are the transmitted symbol voltages. [1]

To determine v_{th}, the conditional probability density functions, $p(v|0)$ and $p(v|1)$ must be matched. The noise is Gaussian and its mean value, μ, adds to the symbol voltages. An equivalent signaling system therefore has symbol voltages of $V_0+\mu$ and $V_1+\mu$, as seen in (3.4.2) and (3.4.3).

$$p(v|0)=\frac{1}{\sigma_0\sqrt{(2\pi)}}.\mathrm{e}^{\frac{-(\mathrm{v_{th}}-(V_0+\mu))^2}{2\sigma_0^2}} \tag{3.4.2}$$

$$p(v|1)=\frac{1}{\sigma_1\sqrt{(2\pi)}}.\mathrm{e}^{\frac{-(\mathrm{v_{th}}-(V_1+\mu))^2}{2\sigma_1^2}}, \tag{3.4.3}$$

where σ_0 and σ_1 are the Gaussian noise standard deviations for the symbols 0 and 1, respectively. For the case in which $\sigma_0=\sigma_1$, the optimum reference is given by (3.4.4).

$$\mathrm{v_{th}}=\frac{(V_1+\mu)^2-(V_0+\mu)^2}{2(V_1-V_0)} \tag{3.4.3}$$

3.4.3 Line Decoding and BER

This block, as the name suggests, will transform the signal (point E), which can have two or three levels and is described by 12 bits, into a signal with two levels which is described only by one bit. In order to do it, the receptor must know the line coding used in the transmission. The decoded signal must resemble as much as possible with the signal transmitted through the communication chain. In the BER (bit error rate) block, the comparison between the original stream and the recovered stream is performed, through MATLAB, in order to determine the number of errors obtained in the transmission. Later, the result can be compared, by the user, with the expected system error probability for the given SNR.

4. Hardware Implementation

Before explaining the hardware scheme used to test the different algorithms used in each block, a brief introduction to the hardware and programming tools will be performed in this chapter.

4.1 Spartan-6 FPGA

The most important part in our system design is the FPGA, namely the Spartan-6 FPGA. A FPGA consists of a large number of gates and flip-flops whose interconnection can be determined, or programmed, after the IC is manufactured [6]. A Configurable Logic Block (CLB) is the main logic block for implementing sequential as well as combinational circuits. A CLB element contains a pair of slices (without direct connectivity between then) and is organized as a column, as depicted in figure 4.1.1.

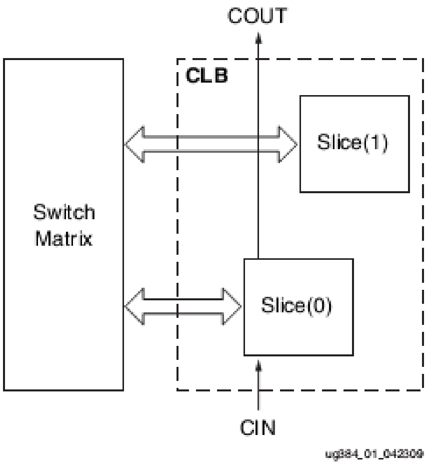


Figure 4.1.1 - Arrangement of Slices within the CLB [7]

Each slice, is further broken down into four logic-function generators (or look-up tables - LUTs) and eight storage elements. The first ones provide logic and ROM functions to the slices, while the second ones acts as memory cells that can be configured as either edge-triggered D-type flip-flops or level-sensitive latches. This leads to 8 LUTs and 16 flip-flops within a CLB. Each slice can be of the type SLICEX, SLICEL, SLICEM. The basic LTUs have 6 inputs and can implement any arbitrarily defined six-input Boolean function. In addition to the basic LUTs, SLICEL and SLICEM contain three multiplexers that are used to combine up to four function generators to provide any function up to seven or eight inputs in a slice.

In the table 4.1.1 is shown the available CLB resources for the Spartan-6 FPGA XC6SLX45 device, used in this project.

Device	Logic Cells	Total Slices	Slices			Number of 6-Input LUTs	Maximum Distributed RAM (kb)	Shift Registers (kb)	Number of Flip-Flops
			SLICEMs	SLICELs	SLICEXs				
XC6SLX45	43,661	6,822	1,602	1,809	3,411	27,288	401	200	54,576

Table 4.1.1 - Logic Resources in one CLB. [7]

This Spartan 6 - CLB's are arranged in a regular array inside the FPGA. Each one connects to a switching matrix for access to the general-purpose routing resources, which run vertically and horizontally between the CLB rows and columns (figure 4.1.2). A similar switching matrix connects other resources such as DSP slices and block RAM. Routing delays vary according to the specific implementation and loading in a design, such as the type of routing, distance required to travel in the device and number of switch matrices to transverse.

The routing (through a place and route tool) of the signal pathways, between the inputs and the outputs of functional elements within the FPGA (IOB's CLB's, DSP slices and block RAM), must then be done in order to deliver optimal system performance and the fastest compile times.

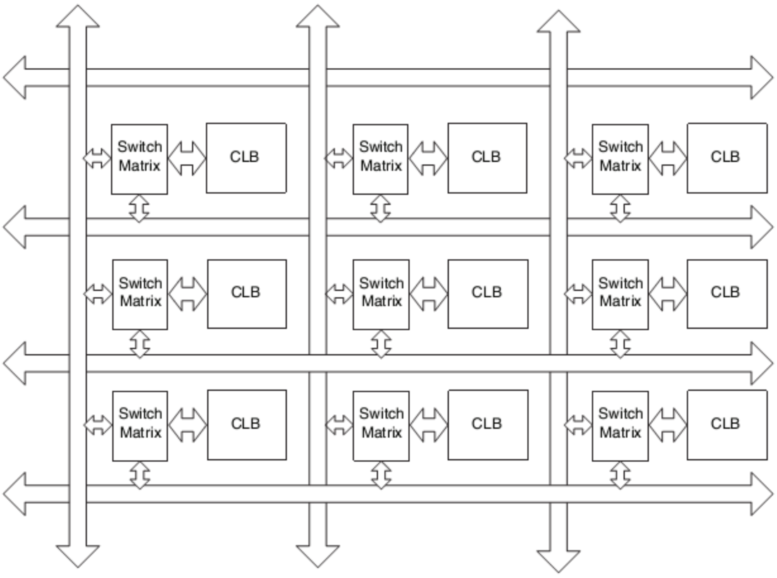


Figure 4.1.2 - CLB Array and Interconnect Channels [7]

4.2 *System Architecture*

As mentioned in section 2.1, the module communicates with an external computer which is running an analyzer implemented in Matlab^{TM}. Since the test ports where the information is read for observation and analysis can vary from experiment to experiment, there must be an entity that coordinates and routes all the available information to the physical output ports in the system (DAC converters). The Embedded Development Kit (EDK) was the tool chosen to perform this work for several reasons:

- It allows a more comprehensive view of the system by separating their blocks (Intellectual Property cores or IP cores) according to the function that they provide.
- Due to the modularity of these blocks, their internal access, insertion or removal in the system becomes easy and enables the integration of custom blocks.
- The SDK (Software Development Kit), an Eclipse open-source framework, allows the C programming language to be used in the MicroBlaze processor configuration, which makes it easier to the MATLAB communication, in comparison with the VHDL programming language.

The EDK provides a suite of design tools that enable us to construct a complete embedded processor system for implementation in a Xilinx FPGA device, including [8]:

- The Xilinx Platform Studio (XPS) Interface - An integrated design environment (GUI) in which it is possible to create the complete embedded design.
- The Embedded System Tools suite;
- Embedded processing IP cores such as processors (also called *pcores*) and peripherals;
- The Platform Studio SDK, which is used to develop the embedded software application.

The embedded software application will run in the MicroBlaze™, an embedded processor soft core that is a reduced instruction set computer (RISC), optimized for implementation in Xilinx® FPGAs. Its soft core processor is highly configurable, allowing the selection of a specific set of features required by the design. The processor fixed feature set, includes [9]:

- 32-bit general purpose registers;
- 32-bit instruction word with three operands and two addressing modes;
- 32-bit address bus.

MicroBlaze is implemented with Harvard memory architecture, in which instruction and data accesses are done in separate address spaces and bus interfaces units. Each address space has a 32-bit range (handles up to 4Gb of instructions and data memory, respectively). In this project, the MicroBlaze is configured with a 32 bit version of the PLB (processor local bus) V4.6 interface, which provides a connection to peripheral and memory access.

MicroBlaze is just one of the system components. Others blocks, such as the local controls, communication chain and UARTs, must be connected in order to the system work as designed. The system is composed by two FPGAs, being the first one responsible for holding the communication chain transmitter and channel parts and the second FPGA having the receiver part (figure 4.2.1).

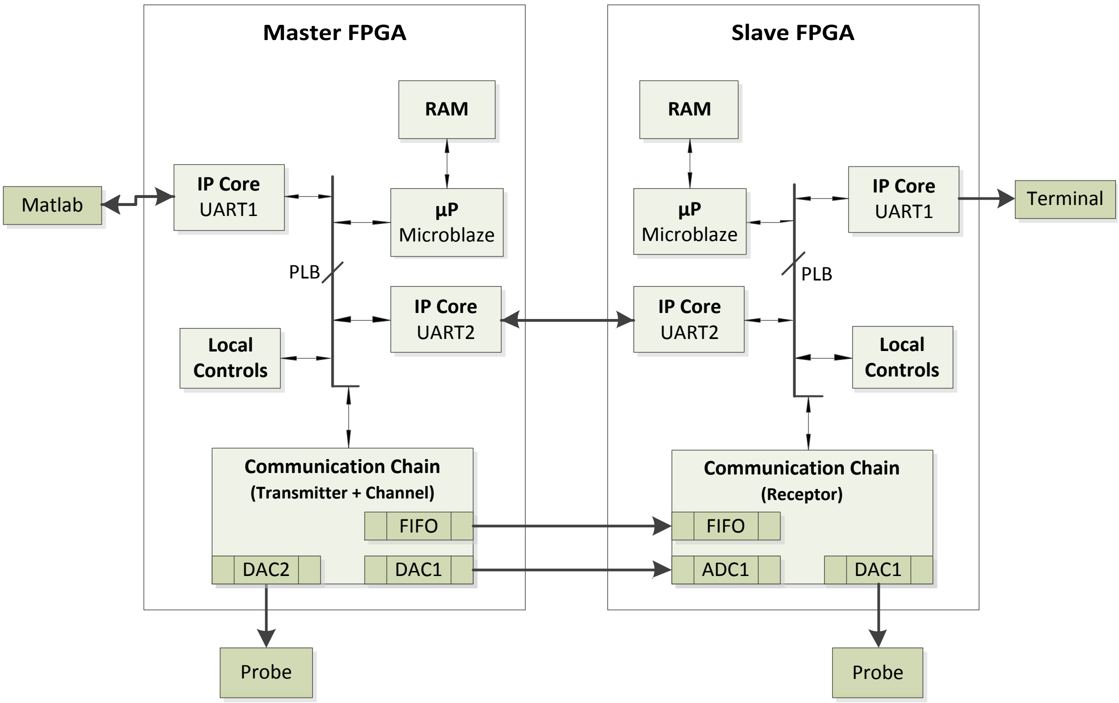


Figure 4.2.1 - System EDK based

By separating the communication chain within the two FPGAs, we impose that the transmitter and the receiver use independent clock generators and therefore approach a real system.

Both FPGAs will communicate to each other through UART2, different from the one used to the communication between the master FPGA and the PC (UART1). The second (or slave) FPGA will be dependent of the master FPGA to communicate with the PC (Matlab) in order to get the initialization values used in the receiver blocks. The UART1 in slave FPGA will not be available for the common user, since it is designed only for tests on the data processed in this FPGA. Each FPGA has also a local control block in each FPGA, which will enable the user to confirm that the FPGA is correctly configured and ready to emulate the system.

The protocols that rule the communication between these two FPGAs and the PC will be described in more detail on chapter five.

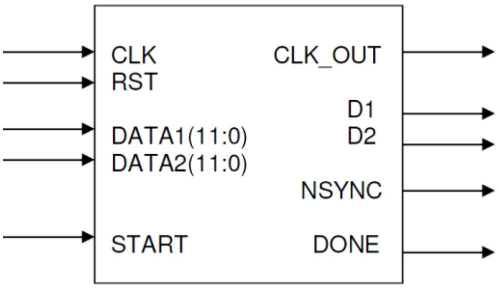
4.3 FPGA Peripherals

As depicted in figure 4.2.1, some peripherals must be connected to each FPGA in order to the project work as designed. In this subchapter, the DAC, ADC and the local controls display will be described.

4.3.1 DAC

Due to their implementation simplicity, resolution and number of channels, the Digilent PmodDA2 Module Converter was the DAC chosen to integrate the system. It converts signals from digital values to analog voltages on two channels simultaneously, with twelve bits of resolution each. The PmodDA2 is powered from the *Digilent* system board connected to it (which holds the FPGA), at 3.3V. Ideally, it will produce an analog output ranging from 0 to 3.3 volts when operating with this power supply voltage [10].

The VHDL component has five inputs and five outputs and has been created by *Digilent* [11]. The input ports are a 50MHz clock (labelled *CLK* and connected to the embedded system clock) that is divided down and used to clock the processes in the component, and a synchronous reset signal (labelled *RST*) that resets the processes which occur inside the component. The data inputs for the two DAC121S101 chips (one for each channel) are two 12-bit vectors (*DATA1* and *DATA2*) that are shifted out serially to the PmodDA2 data pins. The START input signal is used to tell the component when to start a conversion. The output ports of this VHDL component are the divided clock signal *CLK_OUT* (25MHz) and two serial outputs (D1 and D2) that provide the shifted data to the PmodDA2. An *nSYNC* output is used to latch the data inside the PmodDA2 after the data has been shifted out. Lastly, an output labelled *DONE* tells the user when the conversion is done [11]. A block diagram of the VHDL component described above is shown in figure 4.3.1 and the logic that shifts serially the input data word for each channel is depicted in figure 4.3.2. The outputs of this reference component will be connected to the inputs of the PmodDA2.



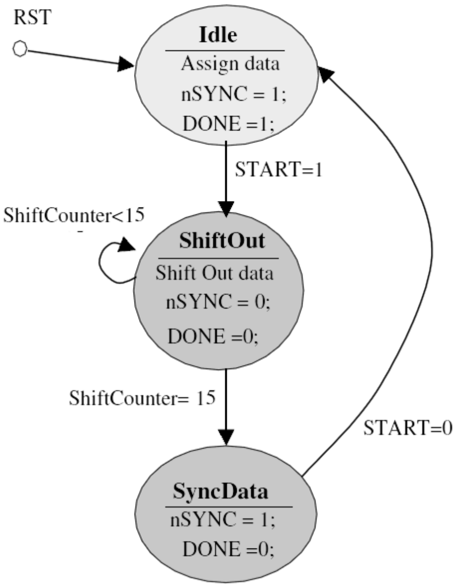


Figure 4.3.1 - PmodDA2 VHDL Reference Component [11]

Figure 4.3.2 - PmodDA2 VHDL Reference Component Finite State Machine [11]

Outside the reference component, the start signal is set high by the time the *DONE* signal is set high by the component. This will enable the user to have a maximum conversion speed in the system. According to the time diagram presented in [11] and the component finite state machine, each DAC121S101 chip will take 17 clock cycles (at 25MHz) to convert a twelve-bit data word. This decreases the rate at which the conversion can be done, for a maximum of 1.47 MHz.

In table 4.3.1 is presented the performance metrics for the DAC block.

Slice Registers		**Slice LUTs**		**Block RAMs**	
Abs.	*%*	*Abs.*	*%*	*Abs.*	*%*
71	0,13	94	0,34	0	0

Table 4.3.1 - Estimate performance metrics for the DAC block mapping

4.3.2 ADC

The chosen ADC, Digilent PmodAD1™ Anallog to Digital Module Converter, like the DAC, has the ability to handle simultaneously two data words with twelve bits each and is powered from the *Digilent* system board connected to it at 3.3V. It will produce a digital output ranging from 0 to 4095 levels, due to the twelve bit precision. The physical module has a 6-pin header and a 6-pin connector [12].

The VHDL component has, like the DAC, 5 inputs and five outputs. The difference to the DAC module, is that now the SCLK output clock that drive each channel (ADC7476 chip) will be 12.5 MHz (instead of the 25 MHz), since the serial bus can run at only up to

20 MHz. A block diagram of the component is shown in figure 4.3.3 and the logic that shifts serially the output data word for each channel is depicted in figure 4.3.4.

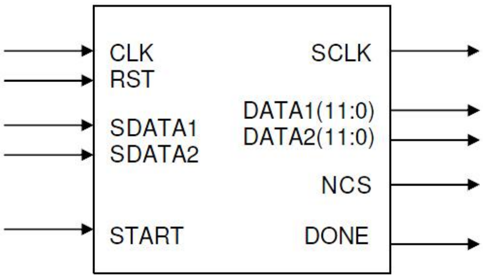


Figure 4.3.3 – PmodAD1 VHDL Reference Component

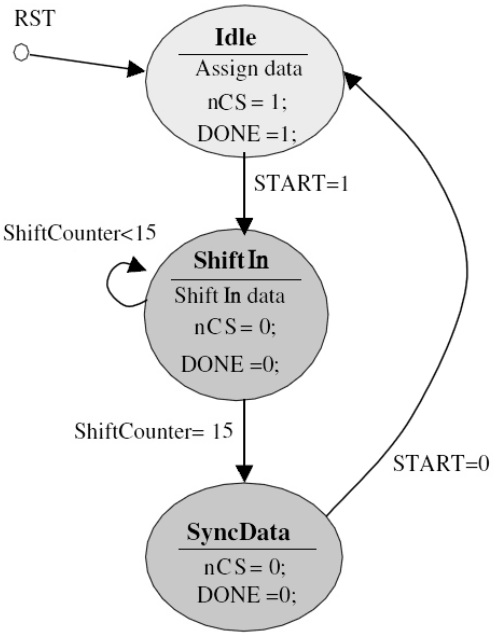


Figure 4.3.4 – PmodAD1 VHDL Reference Component Finite State Machine

The first state is the *IDLE* state in which temporary registers are assigned with the updated value of the input *SDATA1* and *SDATA2*. The next state is the *SHIFTIN* state where the 16-bits of data from each of the ADCS7476 chips are left shifted in the temporary shift registers. The third state, *SYNCDATA,* drives the output signal *nCS* high for 1 clock period maintaining *nCS* high also in the *IDLE* state and telling the ADCS7476 to mark the end of the conversion. There is also a synchronous reset, like in DAC reference component, that will reset all signals to their original state.

Outside the reference component, the *START* signal is set high by the time the *DONE* signal is set high by the component, like the DAC. According to the component finite state machine, each ADCS7476 chip will take 17 clock cycles (at 12,5MHz) to convert a twelve-bit data word. This decreases the rate at which the conversion can be done, for a maximum of 0,735 MHz. Each channel has two 2-pole Sallen-Key anti-alias filters with poles set to 500 kHz. The filters limit the analog signal bandwidth to a frequency range suitable to the sample rate of the converter [12].

In table 4.3.2 is presented the performance metrics for the ADC block.

Slice Registers		**Slice LUTs**		**Block RAMs**	
Abs.	*%*	*Abs.*	*%*	*Abs.*	*%*
72	0,13	85	0,31	0	0

Table 4.3.2 - Estimate performance metrics for the ADC block mapping

4.3.3 Local Control Display

In order to the user know in first instance if the system is operational or not, some sort of visual display is needed. Therefore, an OLED display was used. This device enables to show simple strings of data like the operating frequency, mode of operation, etc. The display information will be sent by the *MicroBlaze* and will be updated according with the Moore state machine, depicted in fig. 4.3.4:

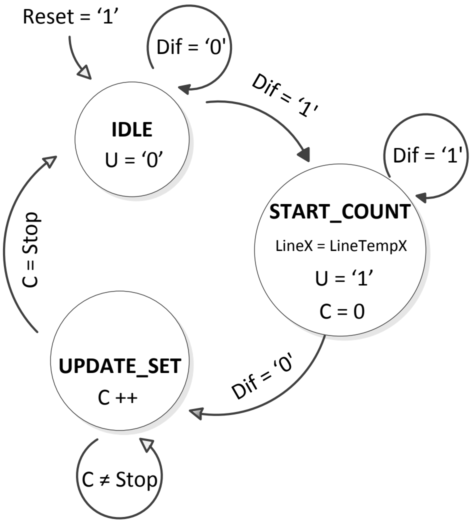


Figure 4.3.4 – Control block state machine

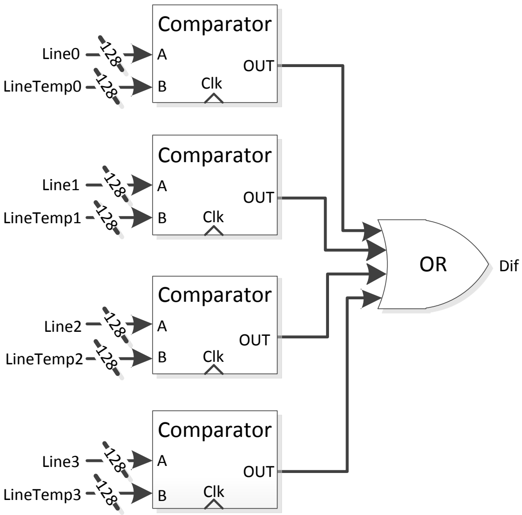


Figure 4.3.5 – *Dif* variable logic scheme

The IDLE state is the first that will be reached, through the asynchronous *Reset* signal. It exits the IDLE state to the START_COUNT state when there is a difference in the future content of the display. This difference is evaluated by the *Dif* variable which changes if any of the data that is sent by the *MicroBlaze* is different from the one is already stored in the display (fig. 4.3.5). As long as the machine is in this state, the *Update* (U) variable is high, which as the name suggests, will update the display content (*LineX*) with the new values stored in the control block variables *LineTempX*. When the *Count* (C) variable reaches a pre-defined value (*Stop*), the IDLE state is reached again and the display is ready for a new update. In table 4.3.3 is presented the performance metrics for the local control block.

Slice Registers		Slice LUTs		Block RAMs	
Abs.	*%*	*Abs.*	*%*	*Abs.*	*%*
175	0,32	259	0,95	1	0,86

Table 4.3.3 - Estimate performance metrics for the local control block mapping

4.4 System Clock

One crucial part of the system is its clock lines. Different blocks in the communication chain need different operation rates, but all the blocks need to be synchronized with each other. In order to ensure this synchronization, the clock rates of all blocks derive from the same major clock line (system clock). In the time diagram taken from a simulation and presented in figure 4.4.1, are depicted the signals that take part on the clock generation process.

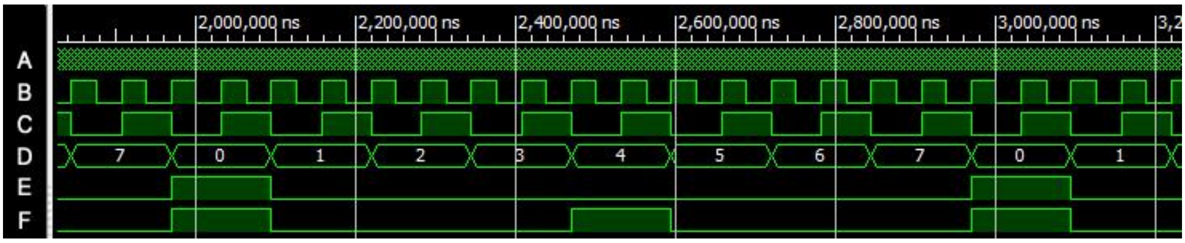


Figure 4.4.1 – System Clock generation signals

First, the 50 MHz embedded system clock ('A' signal), will be divided so that the FIFO block be able to store simultaneously two signals per each sample. Thus, for the system presented in figure 4.4.1, with 8 samples per symbol, the FIFO must store 16 samples (8 per each signal). Its operation frequency, represented by the 'B' signal, will be the higher one in the system after signal 'A'. The programmed/random sequence generator is the only block that changes its output values in a rate 16 times lower (for a sampling rate equal to 8) than the FIFO clock signal ('B'). T he line coding block must be able to change their output value at half the symbol time when the user chooses return-to-zero-like signals. Therefore, the other blocks such as filters, noise and data recover, will use the signal 'C' with half the frequency of 'B' signal, so that they can change their output value by the rate of 8 samples per each generated symbol. In order to the blocks have the same clock, it must be enabled at different time instants. Therefore, the signals 'E' and 'F' were created. The clock enable signal ('E') is set high at the rate of 8 samples per symbol, useful in the sequence generator block. The line coding block will use the 'E' and 'F' clock enabling signals besides the 'C'. The two enable signals 'E' and 'F' are set at the falling edge of the 'C' signal. This ensures that by the time a rising edge occurs in the signal 'C', the enable signals 'E' and 'F' are set. In figure 4.4.2 is depicted an adapted version of figure 3.3.4, with the clock lines for a better understanding of the text above.

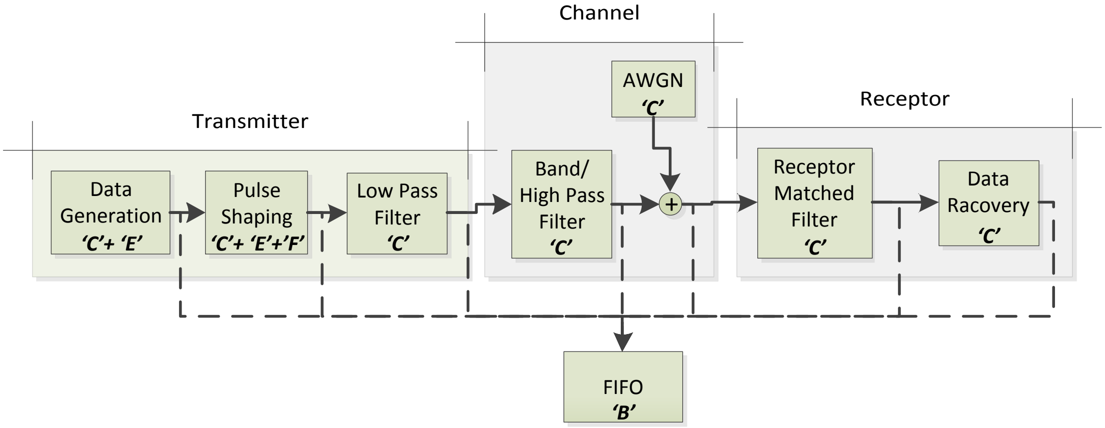


Figure 4.4.2 – System blocks with clock lines

4.5 FIFO

In this project were used two Xilinx IP Cores: the FIFO [13] and the FIR filter [14] (which will be described in section 4.8). The FIFO will allow the data to be sent to Matlab when the system is operating in the *Step-by-Step* mode. The designed FIFO has 32767 positions with 32 bits each. Due to system temporal constrains, it will only be able to store 32752 words of useful data.

The system can be configured to send to Matlab one or two channels. Each channel is stored on alternate positions in the FIFO. If the number of channels is two, the FIFO writing frequency will be twice the sequence generator frequency (as mentioned in the previous 4.4 section) and the maximum number of available words for each channel will be half the total amount of words in the FIFO (16376 words). Therefore, if some signal needs to be analysed with more than 16376 words, the user will have to choose only one channel to fill the FIFO. The way the user will enable each channel and therefore the FIFO operation mode, will be described in appendix A. In the Step-by-Step mode, the clock enable of every block in the communication chain is controlled by the FIFO write enable signal. This way, if for any reason the FIFO stops the writing process (during FIFO filling), no sample will be lost. The FIFO reading process is controlled by the read signal of the FPGA microprocessor and the amount of samples stored in the FIFO, as shown in figure 4.5.1.

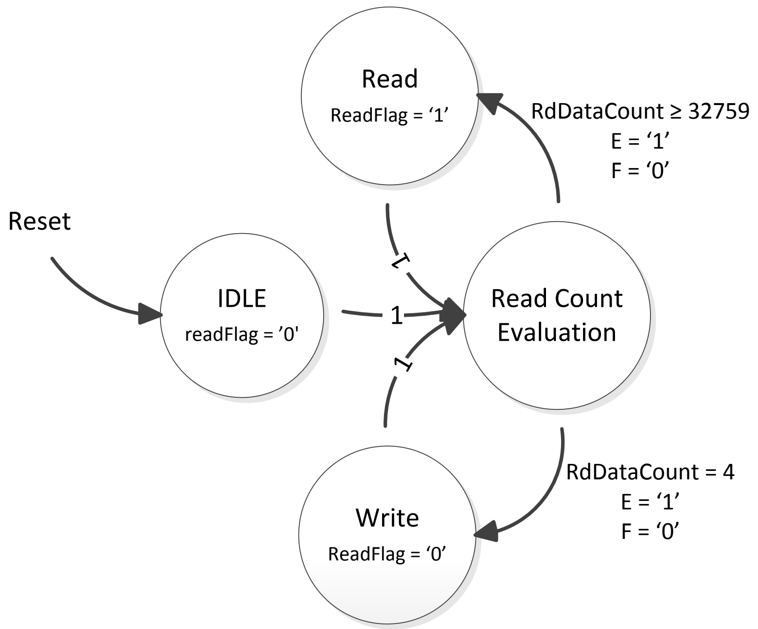


Figure 4.5.1 – Read Flag Flux Diagram

This flux diagram depicts that the microprocessor will only read the samples stored in the FIFO if it has already more than 32759 occupied positions. Furthermore, it will only write on FIFO after a reset has been done, or the number of occupied samples decreases to four after the reading process. These two thresholds were set with the only purpose to avoid the full and empty states of the FIFO. The FIFO top level connections are presented in figure 4.5.2.

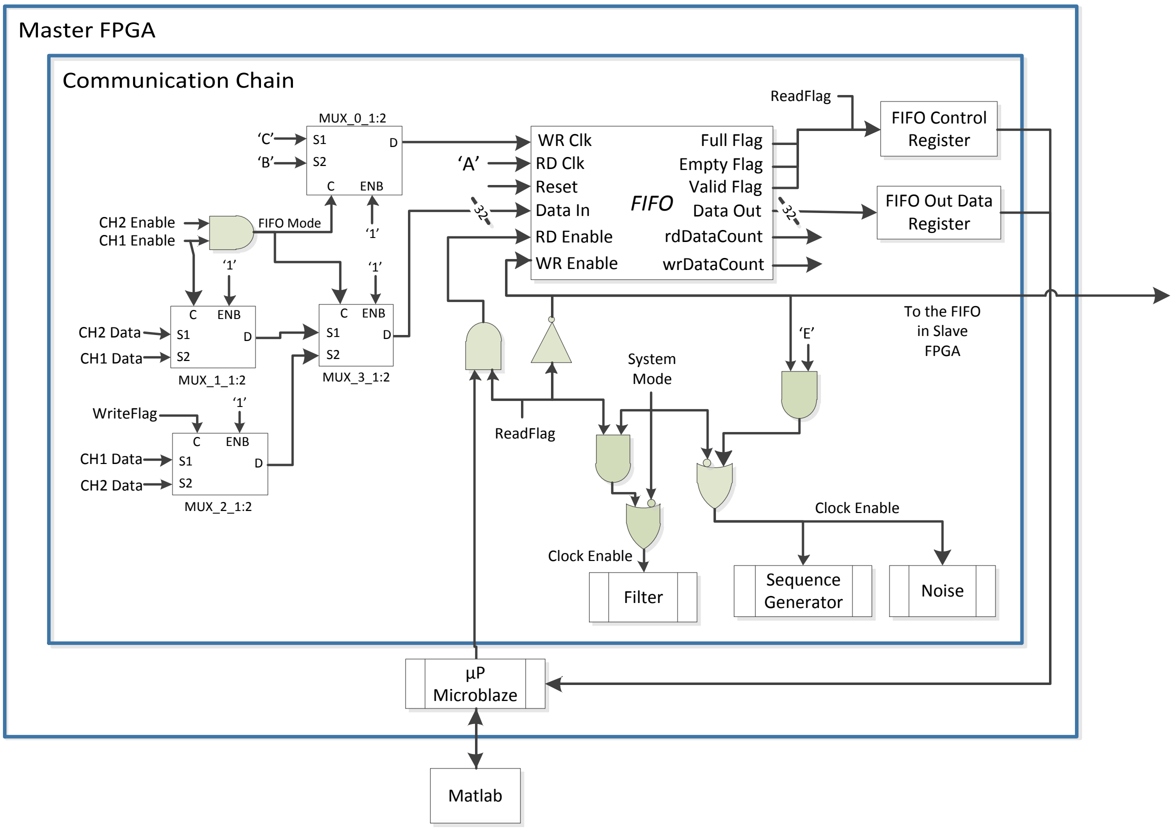


Figure 4.5.2 – FIFO top level connections

In order to the microprocessor know the FIFO current state, one register is available to send the signals *full*, *empty*, *valid* and the *read flag* to the processor. In chapter five will be described in more detail the mechanisms that allow the FIFO reading process.

The approach described in figure 4.5.1 is only valid to the FIFO in the master FPGA. The slave FPGA FIFO write enable is controlled through a dedicated line from master FPGA FIFO (fig. 4.2.1). The slave FIFO will only store values when the master FIFO store, enabling that both FIFOs be synchronized. Therefore, the reading process will be controlled by the complement value of this dedicated line value and the time instants the microprocessor is available for reading. The FIFO mode in each FPGA must be the same. If this equality is not verified by the user, it will be forced by the master FIFO mode.

The FIFO IP Core configuration parameters are presented in appendix B and the estimate performance metrics for this block is presented in table 4.5.1.

Slice Registers		**Slice LUTs**		**Block RAMs**	
Abs.	*%*	*Abs.*	*%*	*Abs.*	*%*
304	0,56	379	1,39	58	50

Table 4.5.1 - Estimate performance metrics for the FIFO block mapping

4.6 Bit Sequence Generator

4.6.1 Pseudorandom Bit Sequence Generator (PRBS)

The PRBS must be implemented for different sequence lengths; that implies that the number of cells needed will vary after the mapping in the FPGA. For example, if the PRBS to be generated has a length 2^4-1 (*n*=4) and later on the user decides to switch to *n*=10, the circuit must be able to make this switch without the need of creating extra hardware.

A generic model must then be created for *n* ϵ [3, 32]. Such a model is presented in Fig. 4.6.1. We can see that, independently of *n*, the number of flip-flops is always equal to 32. The control logic, based on multiplexers, is implemented between each flip-flop (FF) to decide the existence (or not) of feedback (XOR between the FF at the left and the less-significant cell) or a normal right-shift.

As shown in Fig. 4.6.2, the inputs of this block will be the control signals given by the *32x32ROM*, *5:32Decoder* and *32bit Seed*. The first block stores the polynomial accordingly to the number of register cells selected (*#Cells* input). The decoder block is responsible for marking the most significant cell in the register (this cell will be equal to the less significant cell of the register in the next clock). As mentioned in the text above, the *seed* block will be placed in the register in order to define the initial state. Table 4.6.1 shows the truth table of the control logic block, according with the description above.

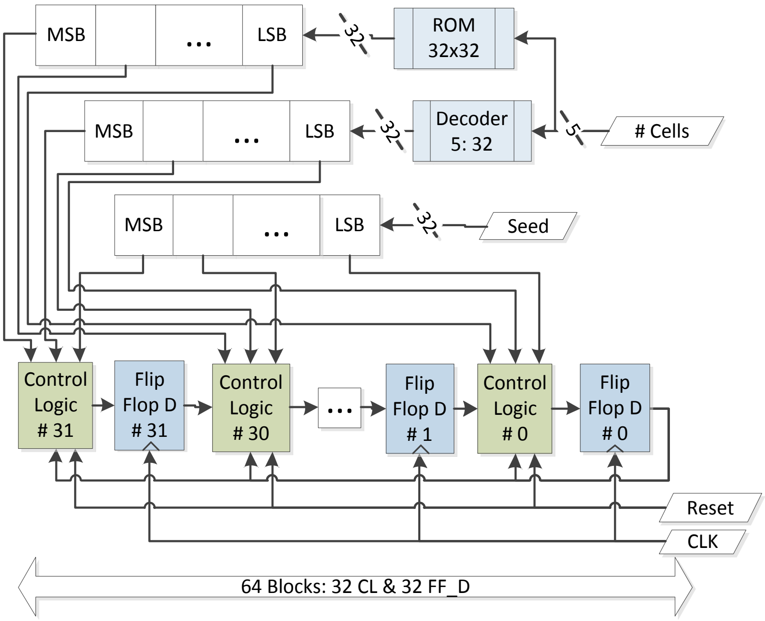


Figure 4.6.1 - PRBS Hardware Implementation

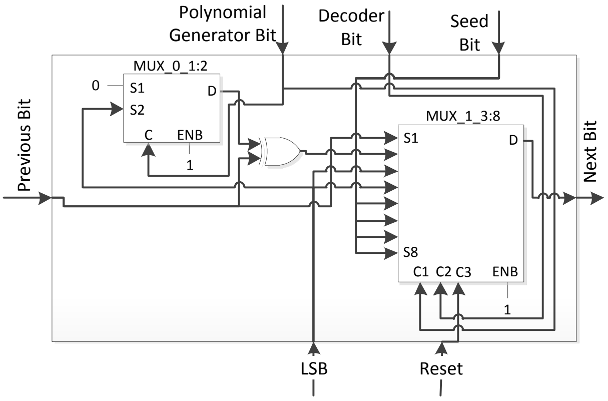


Figure 4.6.2 - Control Logic Block in a PRBS register

Reset bit	Decoder bit	Polynomial bit	Next bit
0	0	0	Keeps the connection
0	0	1	XOR between the LSB and the Previous Bit
0	1	x	LSB
1	x	x	Seed bit

Table 4.6.1 - Control Logic Block Truth Table for PRBS

4.6.2 *Programmed Cyclic Sequence Generator*

The programmed cyclic sequence generator, as mentioned in section 3.1.3, will be the PRBS generator without the feedback taps in the shift register, as depicted in Fig. 4.6.3. Therefore, the control logic block (figure 4.6.4) will not consider the polynomial bit as an input and will have the behavior described in table 4.6.2.

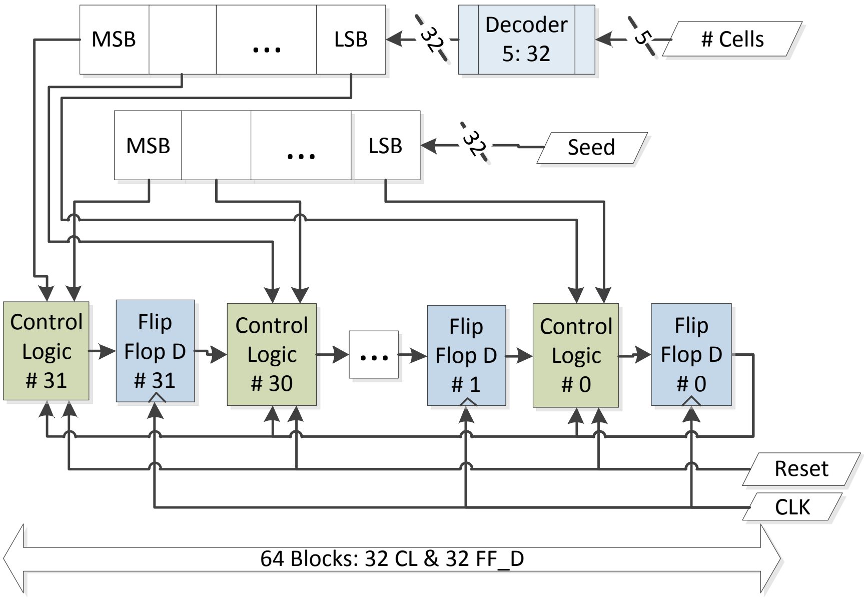


Figure 4.6.3 - Programmed Sequence Hardware Implementation

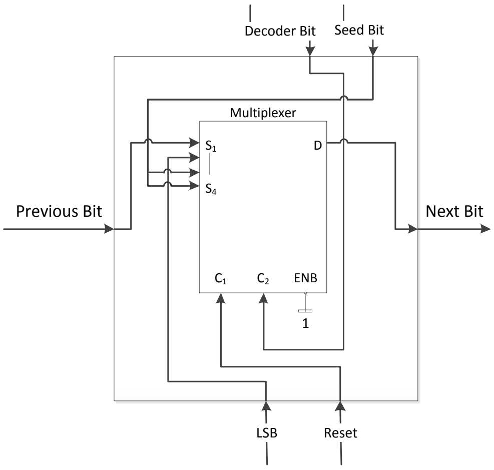


Figure 4.6.4 - Programmed Sequence Logic Control Block

Reset Bit	Decoder Bit	Next Bit
0	0	Previous Bit
0	1	LSB
1	x	Seed Bit

Table 4.6.2 - Control Logic Block Truth table for the Programmed Sequence

4.6.3 *Performance*

In the table 4.6.3 are described the FPGA resources used in the implementation of the sequence generator and the embedded system, where *Abs.* stands for absolute values.

Slice Registers		**Slice LUTs**		**Block RAMs**	
Abs.	*%*	*Abs.*	*%*	*Abs.*	*%*
428	0,78	1510	5,53	0	0

Table 4.6.3 - Estimate performance metrics for the sequence generator mapping

4.7 *Line Coding*

The data output of the cyclic generator is obtained from one of 32 memory cells of the sequence generator register. This data will be coded and sent through the communication chain. Since the line coding result in one of three levels, two bits must be used to code the bit of information. According to the two's complement, the following codification was used:

Line Coding	Voltage Level
"01"	+V
"00"	0
"11"	-V

Table 4.7.1 – Two bit signal codification

Is important to mention that these two bits go throughout the communication chain in parallel, not increasing the bit rate at which the data is sent. In figure 4.7.1 is presented the flux diagram that translates the algorithm for each one of the eight line codes implemented in the system. The line coding output signal may change value by the time a rising edge occurs in the signal clock 'C' (figure 4.4.1). The 'E' and 'F' signals are also referred to figure 4.4.1. The bipolar flag will alternate each time the data bit is equal to '1'.

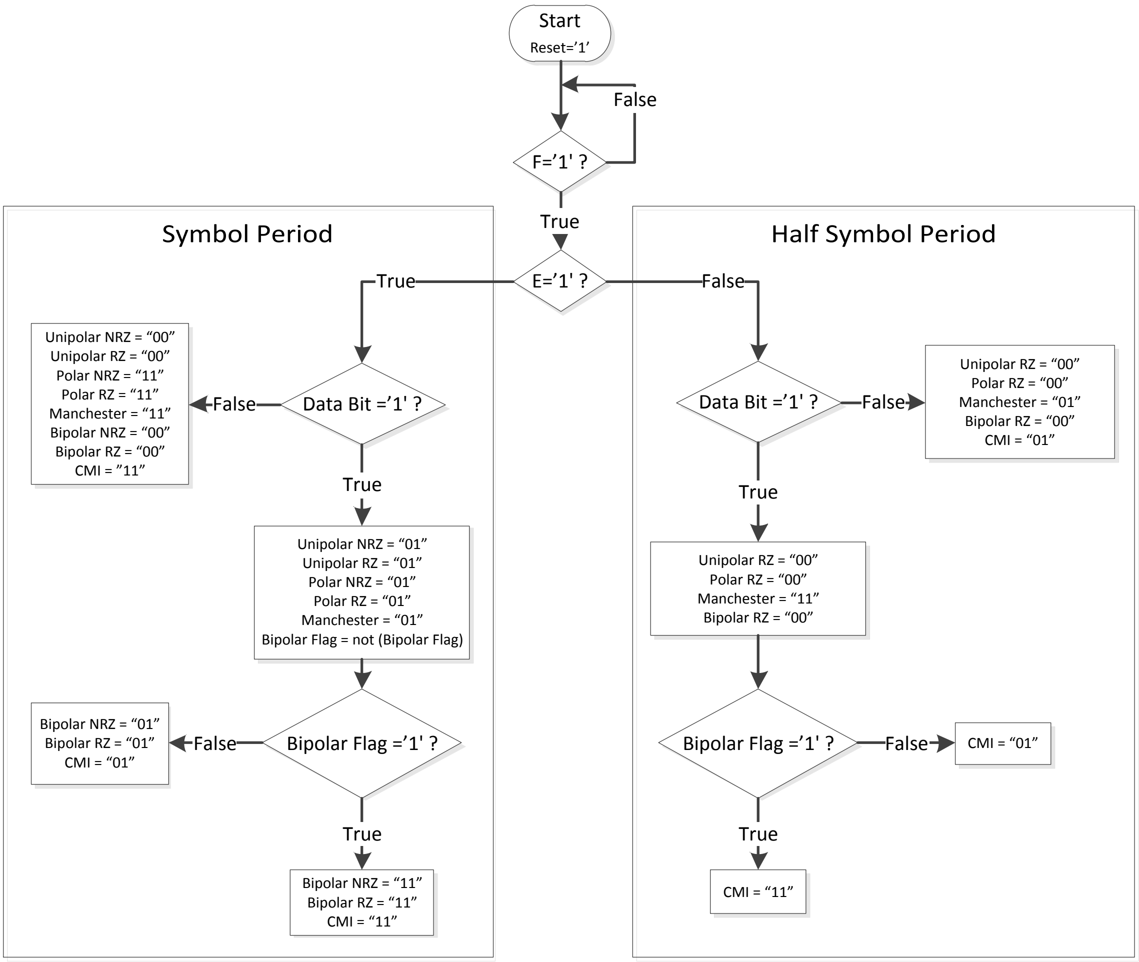


Figure 4.7.1 – Line coding flux diagram

The performance metrics for the line coding block are shown in table 4.7.2

Slice Registers		**Slice LUTs**		**Block RAMs**	
Abs.	*%*	*Abs.*	*%*	*Abs.*	*%*
15	0,03	13	0,04	0	0

Table 4.7.2 - Estimate performance metrics for the line coding block mapping

4.8 *Filters*

As mentioned in chapter three, all the filters used in this project will need to load their coefficients into the FPGA. In order to do that, a Xilinx Fir Compiler IP Core was used, which provides a common interface to generate highly parameterizable, area-efficient high-performance FIR filters [14].

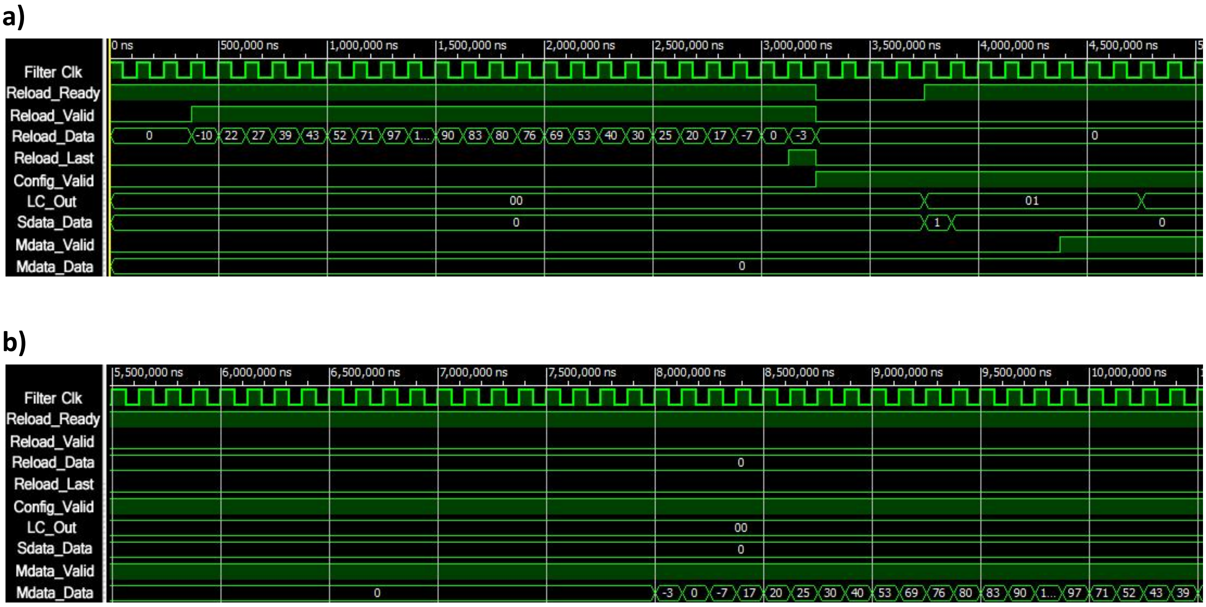


Figure 4.8.1 – FIR Filter data and control signals. Time diagram to (a) and from (b) 5,5 ms

In figure 4.8.1 is shown a simulation time diagram which presents the control and data signals from the FIR filter. For a better understanding of the description below, please refer to the figure B.3a in appendix B, that depicts the filter inputs and outputs. The FIR clock signal, as seen in chapter 4.4, is equal to the symbol clock rate multiplied by the sampling frequency. In time diagram is presented a simulation for a symbol clock rate of 1 kHz and a FIR filter clock equal to 8 kHz (sampling rate equal to eight samples per symbol). The filter output *Reload_Ready* signal will indicate when the filter is ready for loading the coefficients, by setting their value to the logic ‘1’. However, in order to the filter accept the coefficients into the filter, the signal ‘*Reload_Valid*’ must also be high and the signal *Config_Valid* must be at a logic low.

The coefficients are presented in figure 4.8.1 as the *Reload_Data* signal. The filter store 23 signed coefficients 16 bit each and with 12 bits of useful data. To test the filter impulse response, the following random non-symmetric set of 23 coefficients was used: -3, 0, -7, 17, 20, 25, 30, 40, 53, 69, 76, 80, 83, 90, 101, 97, 71, 52, 43, 39, 27, 22, -10, being sent in the reverse order. By the time the last coefficient is sent, the *Reload_Last* signal must be set high only during a filter clock period.

As mentioned in section 4.7, previously to the rised cosine filter (in the transmitter) the data signal is coded, having 2 bits instead of one. After the line coding, this signal (*LC_Out*) will be interpolated, being extracted only one sample, represented here by the *Sdata_Data* signal. The data filter input is signed and have 16 bits with 12 bits of useful information.

The filtered output (signal *Mdata_Data*) will have a valid value, 29 clock cycles after the *Mdata_Valid* signal is high. Due to the coefficient multiplication with the input signal, the *Mdata_Data* signal has 32 bits with 29 useful bits. Since not all the bits are of interest, it was chosen the 12 bit range that best fits the project specifications.

The Estimate performance metrics for each filter created in the system is shown in table 4.8.1

Slice Registers		**Slice LUTs**		**Block RAMs**	
Abs.	*%*	*Abs.*	*%*	*Abs.*	*%*
1,467	2,69	782	2,87	0	0

Table 4.8.1 – Estimate performance metrics for each filter mapping

In the FIR Filter section, in appendix B, is possible to find more information about the configuration parameters of this IP Core.

4.9 AWGN

In this project, the model with K=5 ROM's, was analyzed, as depicted in figure 4.9.1. Each ROM will contain 512 sub-segments of the same length at each level of the partition and each ROM word is composed by 11 bits, in which b=8 bits are from the fractional part.

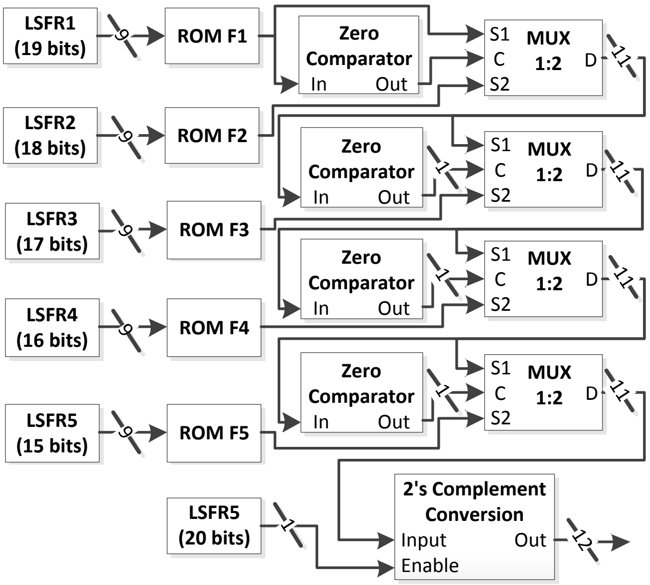


Figure 4.9.1 – AWGN generator schematics

Each ROM will be indexed by different LSFRs with different lengths to ensure that the process is as random as possible. Since each ROM as a length of 512, a minimum of 9 bits must be generated in each LSFR. After each ROM is indexed, their outputs must be prioritized, according to the partitioning level, mentioned in section 3.3.2. For example, the first ROM, containing the values with a low level of partitioning in the interval 0 to 1, will be the first one to be chosen as the output of the system. However, the samples created so far are unsigned. Since we need to run code in FPGA, the arithmetic operations are done in two's – complement, which will reduce the complexity in the design. This last process will add one more bit after the word was chosen. The AWGN generator periodicity will be greatly affected by the periodicity of the signal. This was the reason to use a bigger number of registers in the LSFR for the sign generator (despite only one is used), as depicted in figure 4.9.1.

The estimate performance metrics for the AWGN block is shown in table 4.9.1

Slice Registers		Slice LUTs		Block RAMs	
Abs.	*%*	*Abs.*	*%*	*Abs.*	*%*
137	0,25	241	0,88	6	5,17

Table 4.9.1 – Estimate performance metrics for the AWGN block mapping

4.10 Data Recover

In figure 4.10.1 is depicted a more detailed version of the receptor chain shown in figure 3.4.1.

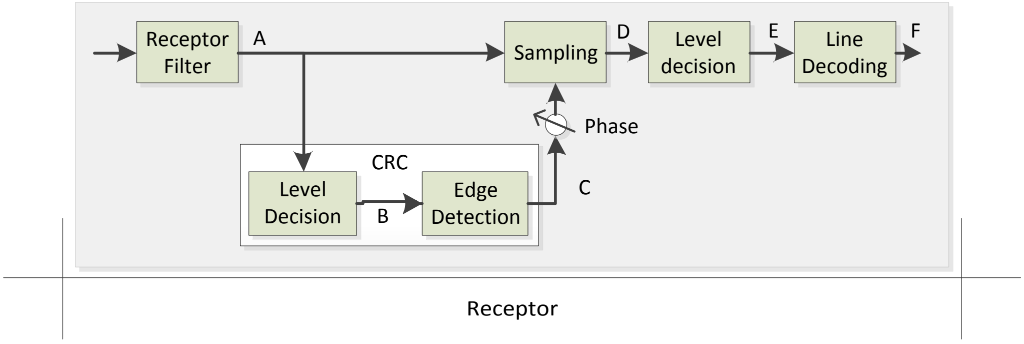


Figure 4.10.1 – Receptor Chain Diagram Block

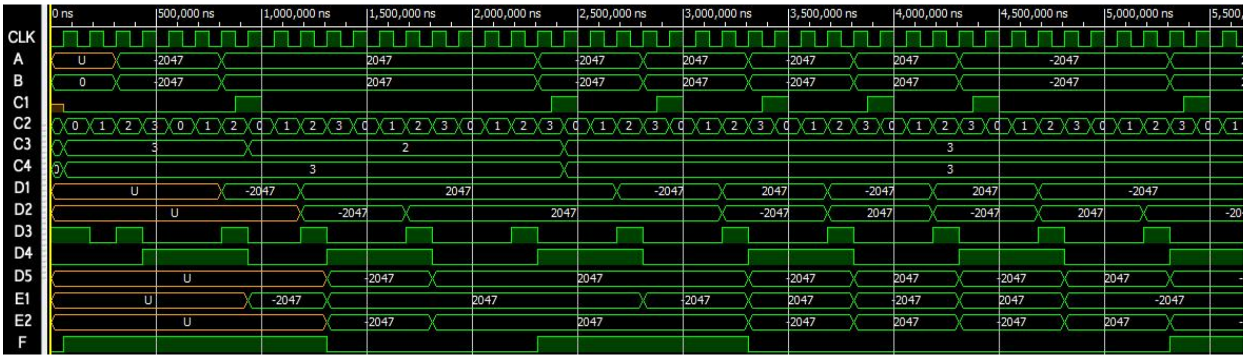


Figure 4.10.2 – Temporal diagram of several test points in the receptor chain. The “U” symbol denotes a non-initialized signal

In fig. 4.10.2 is depicted the result of a test bench simulation for a simple data signal with CMI line coding (point A). The sequence simulated is “01001” with 8 samples per/symbol and an 8 kHz period clock. Some signals used in the VHDL implementation were omitted for reading convenience. Since the ‘A’ signal tested here has only two levels (V+ and V-), the level decision output (B) will result in the same (A) signal. This would not happen if for example the A signal is from the filter output, which can have 4096 levels.

The signal in C1 represents the edge detection extracted through XOR logic between the B signal and a clock period delay of this same signal. This C1 signal was set to be able to change value in each clock falling-edge so that the count (C2) could change in the next clock rising-edge. The maximum count of signal C2 is a function of the C1 transitions, the pre-defined sampling rate and the previous count values stored in memory. In fig. 4.10.3 flux diagram is shown the dependency between signals C1, C2, C3 and C4. These four signals are clock dependent.

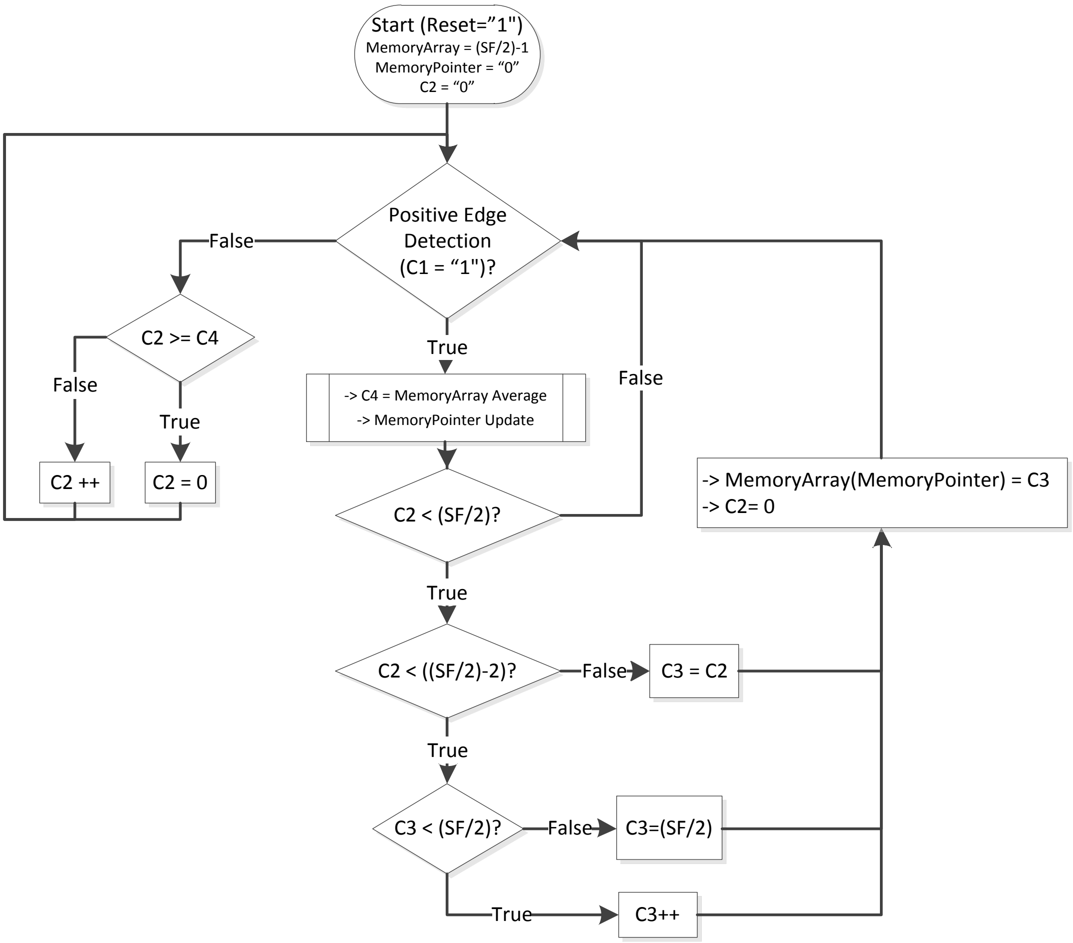


Figure 4.10.3 – C point flux diagram – rising-edge clock dependent

This algorithm must be able to detect two transitions per symbol, as happens in return to zero signals, and to accommodate data signal variations due to jitter and due to bad level decision. Therefore, the count must take one of the three following values: $\left\{\frac{SF}{2}-2,\frac{SF}{2}-1,\frac{SF}{2}\right\}$, where SF denotes the sampling factor. In this case, where SF=8, the ideal count limit is $\frac{SF}{2}-1=3$, since the count start on the 0 value. In order to achieve a better resistance to noise, an average value will be calculated between the most recent count and the previous ones. A new count value will be stored in the next position (given by memory pointer) of the memory array, only when occurs an edge detection (C1). The memory pointer will then be updated. This memory array stores the values used in the average and can take the sizes 1, 2, 4, 8, 16 and 32, which is chosen by the user. In fig. 4.10.2, C3 is the most recent value placed in a memory array with 8 positions. C4 is the average of all the eight values placed in memory array. Due to design restrictions, at least half clock period delay occurs between the instant changes of the signals C3 and C4.

The sample instant is obtained adding the *Phase* parameter to the C2 signal. This parameter is set by the user and can take any value between 0 and $\frac{SF}{2} - 1$. In the example that we are analyzing, the sampling instant was set to 1, this is, *Phase*=1. Signal D1 is the sampled signal and D2 is D1 delayed half symbol period. This two signals will allow to obtain the D3 signal, which is another edge detection, but now, after sampling and repeated when C2=Phase. On the other hand, the clock signal (D4) will need to analyze three signals: D1, D2 and D3, in order to know when the clock takes the "1" or "0" level. For example, in the case of the CMI signal, if D3=1 and D1 and D2 are different, we are sure that the clock will be equal to '1'. Otherwise, if D3=1 and D1 and D2 are equal, the clock will change value independently of D1 and D2 (as it happens in a "1" transmission). D3 is delayed one clock cycle in comparison with the actual sampling instant given by C2=Phase.

Signal D5 is taken before the decision is made, and represents the A signal sampled according to the instants given by D3. D5 is delayed (SF/2) +1 clock cycles from the sampling instant, in order to be compared directly, by the user, with the decoded F signal.

The signal E1 is the result of the second level decision made in the receptor chain. This decision uses the same upper and lower thresholds as the first level decision made in the CRC block. The difference is that in this second case, the input signal is the sampled signal (D1) instead of the receptor filter output signal (A). E1 signal will be delayed one clock cycle from D1. E2 will be delayed one clock cycle from D2.

At point F we will have the decoded data, which must resemble as much as possible with the original data transmitted in the chain (by the sequence generator at the transmitter). The delay between the point A and point F will depend on the selected line code on the transmitter. For a CMI line coding, the delay between this two points will be half symbol period bigger that the other codes, because in the design used, the decoding (F) will need to get the information from the next half period (E1) in order to decode the actual signal (E2), which may resemble like a non-casual process. The flux diagram in fig. 4.10.4 shows the algorithm used to decode the data for each one of the 8 codes used.

The estimate performance metrics for all the data recover block is shown in table 4.10.1

Slice Registers		**Slice LUTs**		**Block RAMs**	
Abs.	*%*	*Abs.*	*%*	*Abs.*	*%*
209	0,38	330	1,20	0	0

Table 4.10.1 – Estimate performance metrics for data recovery mapping

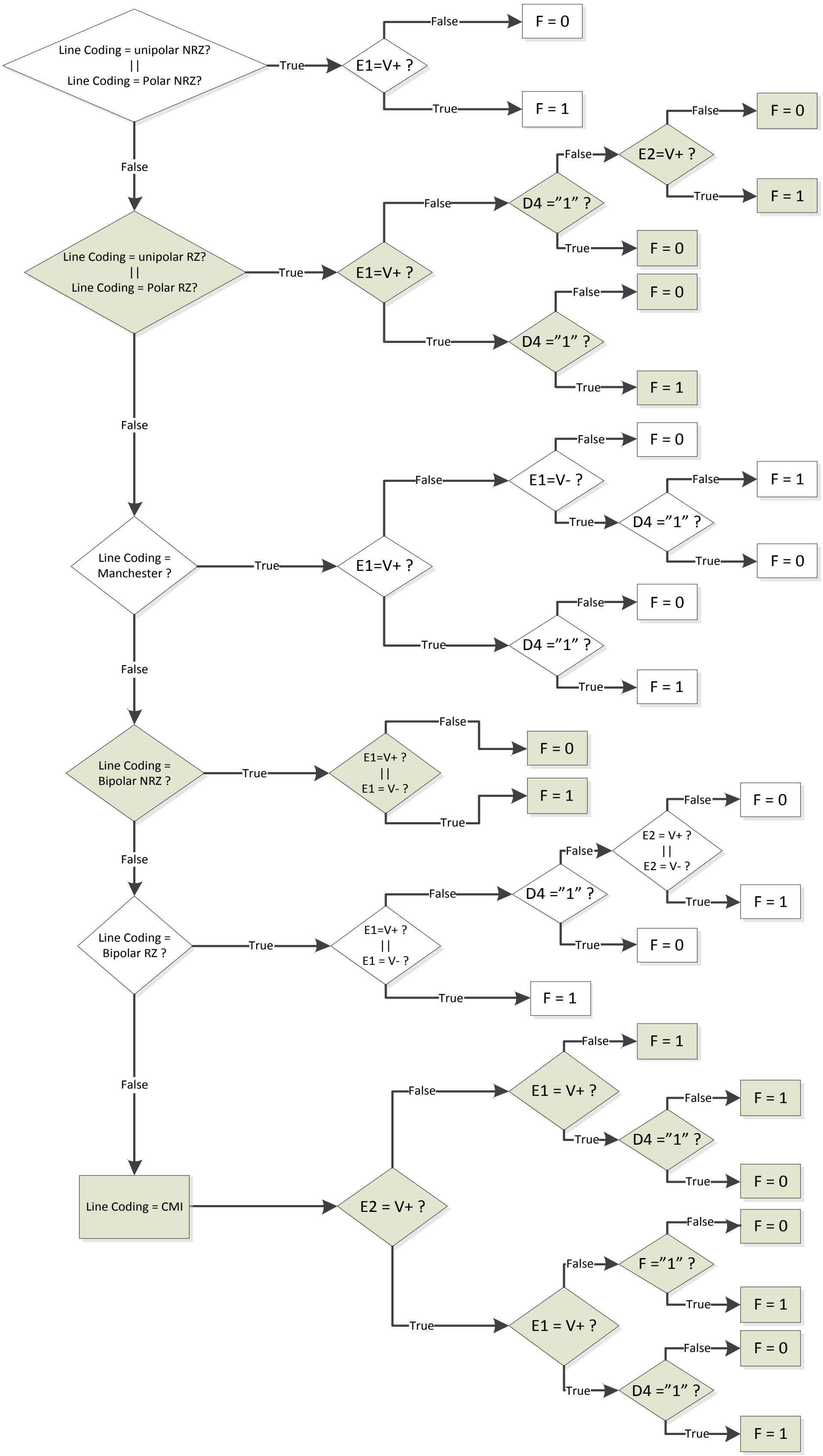


Figure 4.10.4 – Line decoding flux diagram - Clock independent behavioral model

5. *Software*

The MicroBlaze® microprocessor is an important part of the embedded system, since it will be the bridge between the PC and the communication chain, as it can be seen in figure 4.2.1. In this chapter, will be described the mechanisms that rule the data exchange between these two entities.

5.1 Block Initialization

As mentioned in chapter four, all blocks in the chain have a group of parameters that need to be set before they can generate or process data. The data of these parameters come from the PC in the form of commands. The master FPGA processor will first receive the commands and route them (or not) to the slave FPGA. The commands (or part of them) will return back to the source as an echo, so that the user may know if they were correctly sent to both FPGAs, as depicted in figure 5.1.1 temporal diagram. Is important to mention that in the case when the command is sent to the slave FPGA, the echo to the PC (Matlab analyzer) will only be done when the terminator '\r' is received from this slave FPGA.

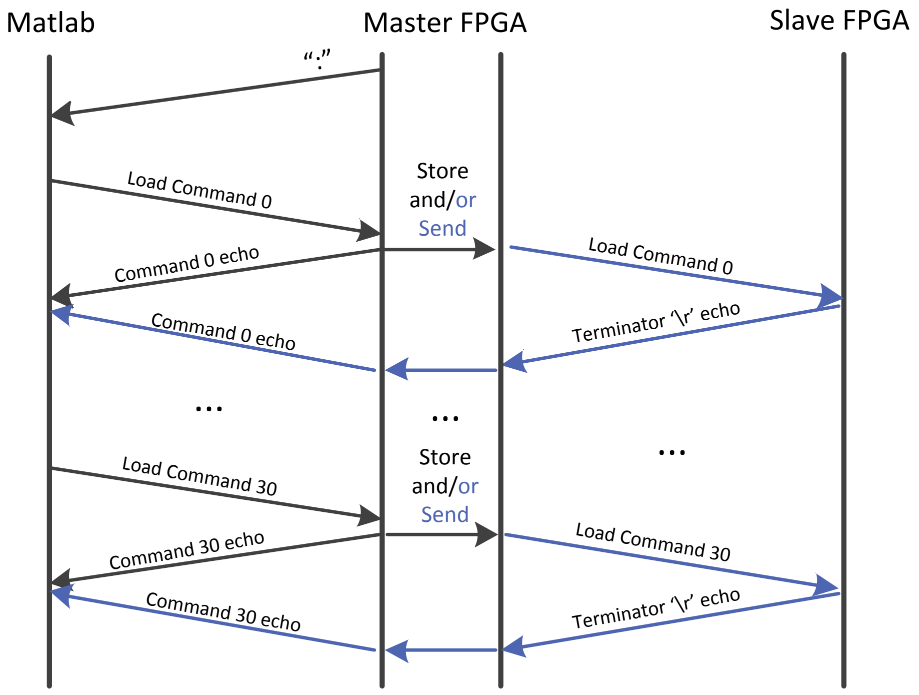


Figure 5.1.1 – Block initialization temporal diagram

A command is composed by the following structure: ">ABC.DE.Data\r" where ABC corresponds to the block place in the chain and DE corresponds to the parameter of that block, being each letter a byte. The “Data” component size will depend on the type of parameter and will be in most cases, digits from 0 to 9. In order to processor know when the command is ended, a terminator ‘\r’ (Enter) is the last command component. More info about the command structure can be found in appendix A of this document. Figure 5.1.2 presents the flux diagram of each command load in the master FPGA.

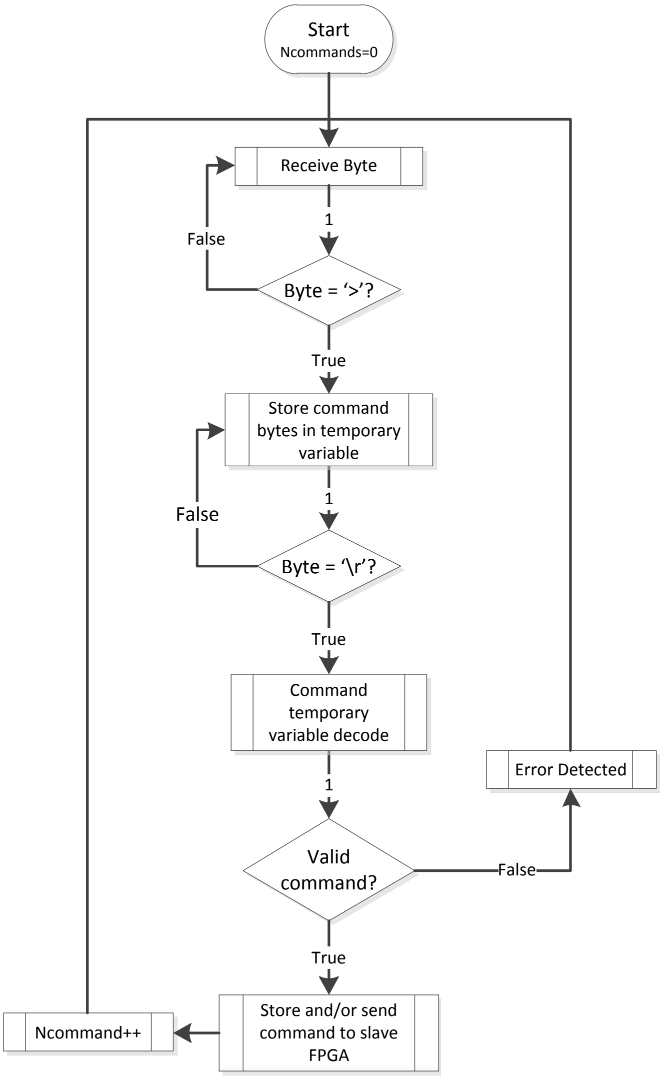
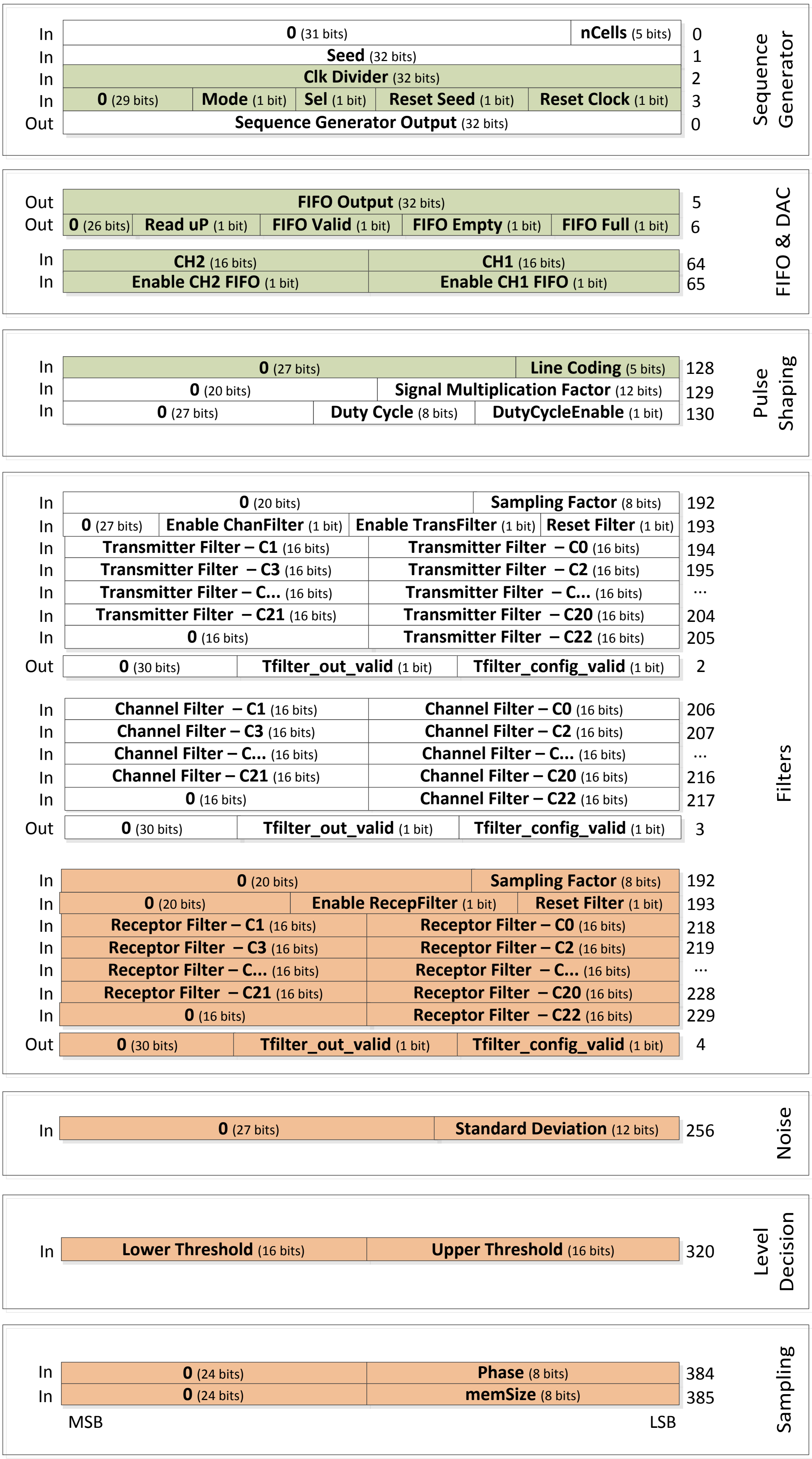


Figure 5.1.2 – Command Load flux diagram in master FPGA

The “Valid command” confirmation is just a state machine in which the several command components are consecutively tested to check if it corresponds to one of the existing parameters in the system. After validation the command can be either stored in the master FPGA registers and/or sent to the slave FPGA so that this last one component can do the same procedure. This choice depends on the type of parameter that the given command comprehends. In table A.1.1 is possible to see which commands exist in the system and to which FPGAs they belong. For example, the system clock parameter will have to be sent to both FPGAs while the transmitter filter coefficients only concern to the master FPGA.

All IP Cores created in Embedded Development Kit (EDK), like the communication chain, have a fixed set of registers that can be used to communicate to/from the processor. However, if the project needs more registers, a new IP core needs to be created to accommodate this change and the previous code must be transferred to the newly created IP Core. Therefore, this system limitation in numbers of registers becomes non practical. To avoid it, RAM memories were designed in VHDL, with a size of 320 and 448 positions in the master and slave FPGA, respectively. The communication chain IP core was created with 32 registers. The data that comes from the processor will be first stored in register 31 and its position in the RAM will be stored in register 30. On the other hand, the information that goes to the processor will be stored directly in the IP core registers, so the access to the information by the processor will be faster. All the communication chain registers and RAM words are presented in figure 5.1.3, having 32 bits each and being the right bit, the least significant bit (LSB).



Sequence Generator

In	0 (31 bits)	nCells (5 bits)				0
In	Seed (32 bits)					1
In	Clk Divider (32 bits)					2
In	0 (29 bits)	Mode (1 bit)	Sel (1 bit)	Reset Seed (1 bit)	Reset Clock (1 bit)	3
Out	Sequence Generator Output (32 bits)					0

FIFO & DAC

Out	FIFO Output (32 bits)					5
Out	0 (26 bits)	Read uP (1 bit)	FIFO Valid (1 bit)	FIFO Empty (1 bit)	FIFO Full (1 bit)	6
In	CH2 (16 bits)	CH1 (16 bits)				64
In	Enable CH2 FIFO (1 bit)	Enable CH1 FIFO (1 bit)				65

Pulse Shaping

In	0 (27 bits)	Line Coding (5 bits)		128
In	0 (20 bits)	Signal Multiplication Factor (12 bits)		129
In	0 (27 bits)	Duty Cycle (8 bits)	DutyCycleEnable (1 bit)	130

Filters

In	0 (20 bits)	Sampling Factor (8 bits)			192
In	0 (27 bits)	Enable ChanFilter (1 bit)	Enable TransFilter (1 bit)	Reset Filter (1 bit)	193
In	Transmitter Filter – C1 (16 bits)	Transmitter Filter – C0 (16 bits)			194
In	Transmitter Filter – C3 (16 bits)	Transmitter Filter – C2 (16 bits)			195
In	Transmitter Filter – C... (16 bits)	Transmitter Filter – C... (16 bits)			...
In	Transmitter Filter – C21 (16 bits)	Transmitter Filter – C20 (16 bits)			204
In	0 (16 bits)	Transmitter Filter – C22 (16 bits)			205
Out	0 (30 bits)	Tfilter_out_valid (1 bit)	Tfilter_config_valid (1 bit)		2
In	Channel Filter – C1 (16 bits)	Channel Filter – C0 (16 bits)			206
In	Channel Filter – C3 (16 bits)	Channel Filter – C2 (16 bits)			207
In	Channel Filter – C... (16 bits)	Channel Filter – C... (16 bits)			...
In	Channel Filter – C21 (16 bits)	Channel Filter – C20 (16 bits)			216
In	0 (16 bits)	Channel Filter – C22 (16 bits)			217
Out	0 (30 bits)	Tfilter_out_valid (1 bit)	Tfilter_config_valid (1 bit)		3
In	0 (20 bits)	Sampling Factor (8 bits)			192
In	0 (20 bits)	Enable RecepFilter (1 bit)	Reset Filter (1 bit)		193
In	Receptor Filter – C1 (16 bits)	Receptor Filter – C0 (16 bits)			218
In	Receptor Filter – C3 (16 bits)	Receptor Filter – C2 (16 bits)			219
In	Receptor Filter – C... (16 bits)	Receptor Filter – C... (16 bits)			...
In	Receptor Filter – C21 (16 bits)	Receptor Filter – C20 (16 bits)			228
In	0 (16 bits)	Receptor Filter – C22 (16 bits)			229
Out	0 (30 bits)	Tfilter_out_valid (1 bit)	Tfilter_config_valid (1 bit)		4

Noise

In	0 (27 bits)	Standard Deviation (12 bits)	256

Level Decision

In	Lower Threshold (16 bits)	Upper Threshold (16 bits)	320

Sampling

In	0 (24 bits)	Phase (8 bits)	384
In	0 (24 bits)	memSize (8 bits)	385
	MSB	LSB	

Figure 5.1.3 – Communication Chain Data words

The green filled boxes represent words common to both FPGAs, the white color correspond only to master FPGA and the orange color to the salve FPGA. The words that will be sent to the processor are represented by the *Out* label and data that comes from the processor and will be stored in the RAM is represented by the *In* label. For each type of data word division, such as the sequence generator, pulse shaping, etc, there is an addresses jump of 64 positions, in order to address more words in those divisions in the future.

5.2 Data Sending

After all the blocks in the system are initialized, they can start to generate/process data. As mentioned in section 4.2, in order to read that data, is possible to use either a FIFO or a DAC. If the DAC is selected (*Continuous* mode) a probe must be plugged into the system, like an oscilloscope. But if the user wants to analyze the data in the PC in a more accurate way, the *Step-by-Step* mode must be selected. The FIFO is used in this last mode, requiring an interaction algorithm between both FPGAs and the PC. This algorithm is shown in figure 5.1.4. The number of words that the user specifies in the command is the number of words for two channels. Therefore, both FPGAs will send the double number of words specified by the user, being this number the stop condition in the flux diagram in figure 5.1.4. The data from each FPGA will be sent to the PC in an alternated way. A handshake must be done between the master and the slave FPGA in order to know if the information that the slave is sending, corresponds to the data word. This handshake is done with the bytes '>' and '<' that limit the start and the end of the data word. Due to software constrains, in the slave to the master FPGA communication, it can only be sent one byte at a time. Since the data word has 32 bits, it must be splitted in four pieces in the slave FPGA and reconstructed in the master FPGA. This process is represented in the box "Data Word Receive" in figure 5.1.4.

In figure 5.1.5 is represented the flux diagram of the algorithm used to send the data stored in the slave FPGA FIFO to the master FPGA. This last diagram is a soft version of the master FPGA flux diagram.

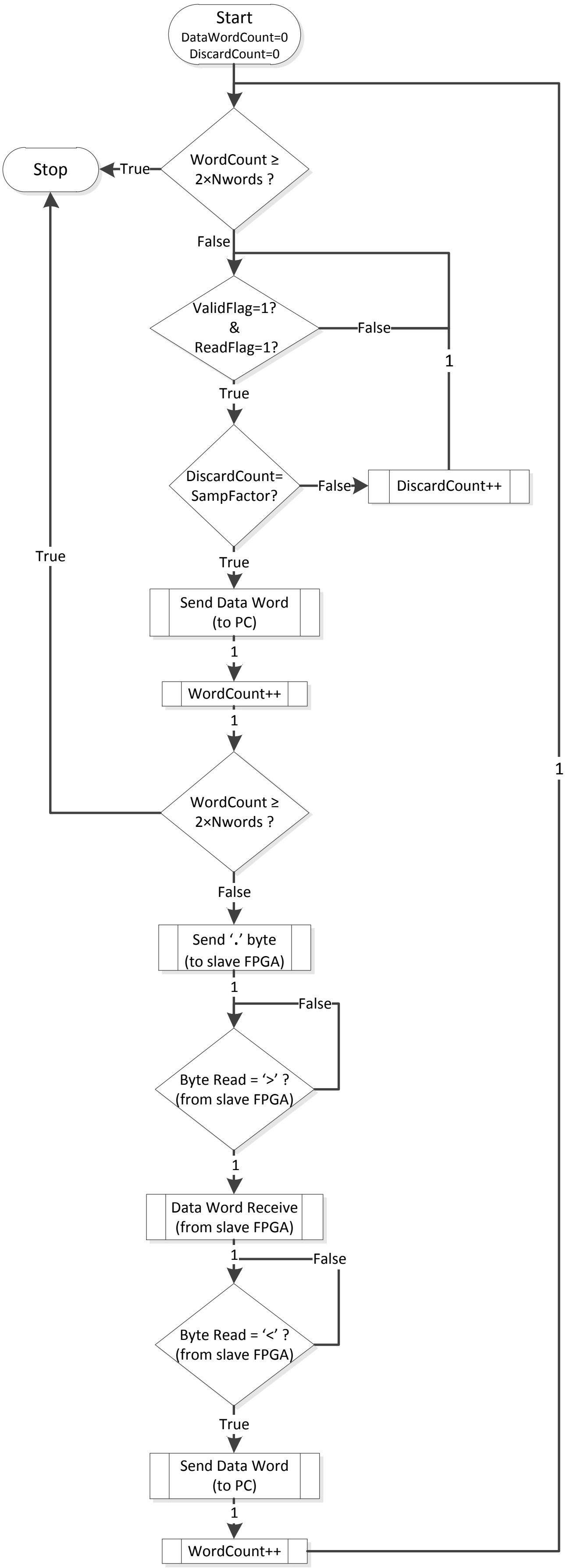

Start
DataWordCount=0
DiscardCount=0
WordCount ≥ 2×Nwords ?
True
Stop
False
ValidFlag=1? & ReadFlag=1?
False
1
True
DiscardCount= SampFactor?
False
DiscardCount++
True
True
Send Data Word (to PC)
1
WordCount++
1
WordCount ≥ 2×Nwords ?
False
Send '.' byte (to slave FPGA)
1
False
Byte Read = '>' ? (from slave FPGA)
1
Data Word Receive (from slave FPGA)
1
False
Byte Read = '<' ? (from slave FPGA)
True
Send Data Word (to PC)
1
WordCount++
1

Figure 5.1.4 – Master FPGA data sending flux diagram

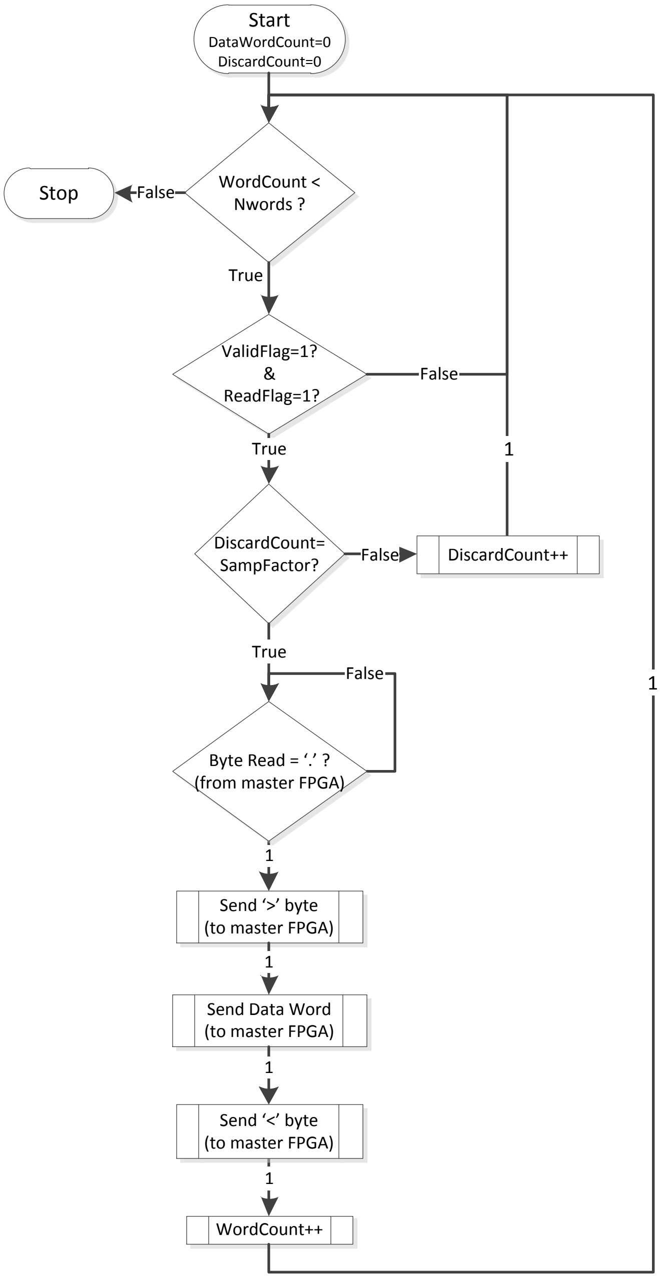


Figure 5.1.5 – Slave FPGA data sending flux diagram

6. Experimental Results

6.1 Sequence Data Generator

As mentioned in section 4.6, the sequence data generator enters data into the system in the form of a programmed sequence or a random sequence. The system will work at $R_s = 10k\ symbols/s$ and a sampling factor of $F_s = 8\ samples/symbols$. In order to test the performance of this block, it was generated 56 samples of 32 bits each, through a random sequence of length $2^3 - 1 = 7$. The results of the created Matlab analyser are shown in figure 6.1.1.

a) b)

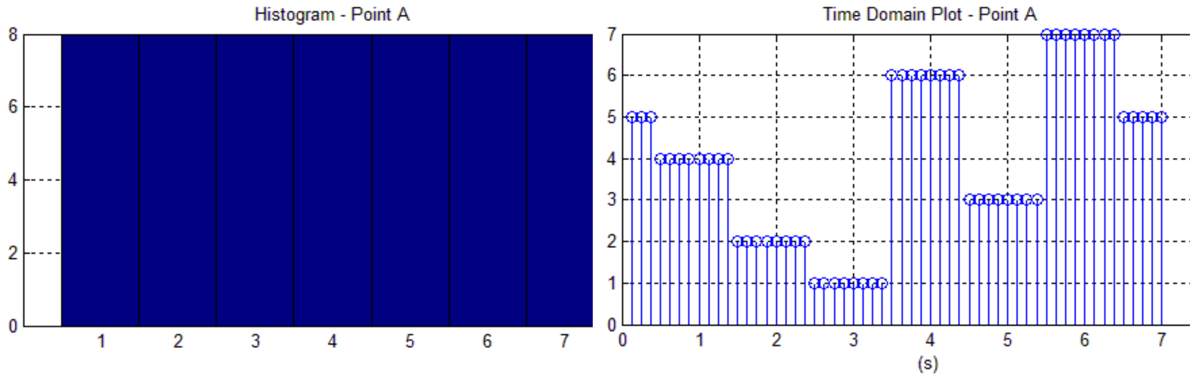


Figure 6.1.1 – PRBS generator output experiment. a) histogram and b) time domain plot

It is possible to observe from this example that each generated symbol was sampled 8 times since 56 samples correspond exactly to $8\ samples/symbol \times 7 symbols$.

Now changing the PRBS to the programmed sequence generator and maintaining the rest of the previous parameters, the figure 6.1.2 is obtained. The chosen seed has now as important role, since it will define the values which the generator can take through its shifting process. In both generators the seed was set to five. For a generator with three cells, it can take the values {5,6,3} since the three initial bits "101" will suffer a right shift for each symbol clock period, T_S. The sequence will then repeat itself after 3 symbols (or 18 samples).

a) b)

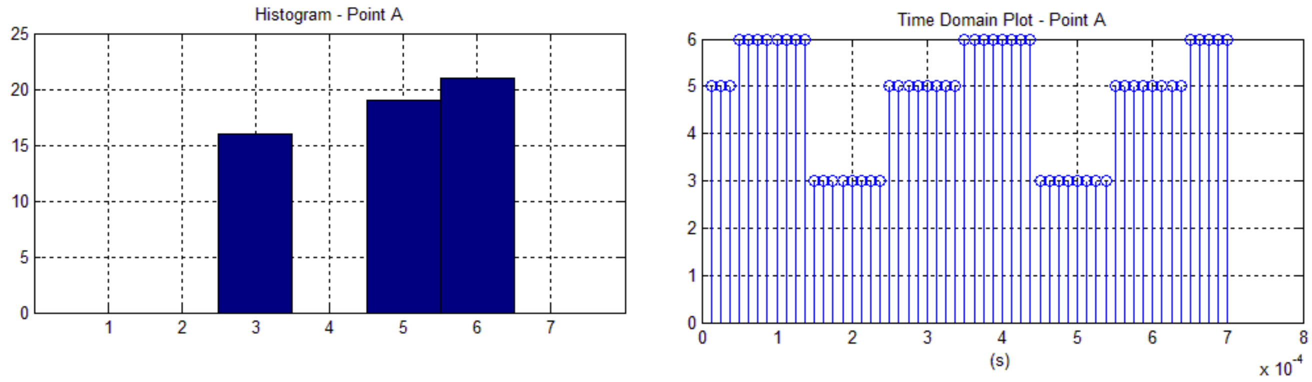


Figure 6.1.2 – Programmed generator output experiment. a) histogram and b) time domain plot

6.2 *Pulse Shaping*

From this point on, only the least significant bit (from the 32 bits) of the sequence generator output will be considered. Therefore, if the generated data word with 32 bits is an odd number (in decimal base), the transmitted bit in the chain will be equal to one. Otherwise, for an even number, the bit will be zero. Like in the case of the sequence generator, in order to test the designed pulse shaping blocks, will be used a symbol (bit) rate of $R_s = 10k\ symbols/s$ and a sampling factor of $8\ samples/symbol$.

6.2.1 *Line coding*

The line coding transforms the binary signal into a 2 bit signal that will be then extended to a 12 bit signal in which the highest level is $V^+ = 2047$ and the lowest level is $V^- = -2047$. Therefore, each data signal sample after the line coding block can have 4096 levels. However, before the PSD generation, each data sample was normalized to $V^+ = 2047$.

6.2.1.1 Deterministic Sequence

Considering the generation of a programmed sequence with 15 bits, in which the first 5 bits are set to ‘1’ and the other 10 bits are set to ‘0’. The correspondent seed is equal to the decimal translation of the binary word $111110000000000_B = 31744_{10}$. Considering first the unipolar NRZ signal, the resulting PSD for the conditions mentioned above is presented in figure 6.2.1.

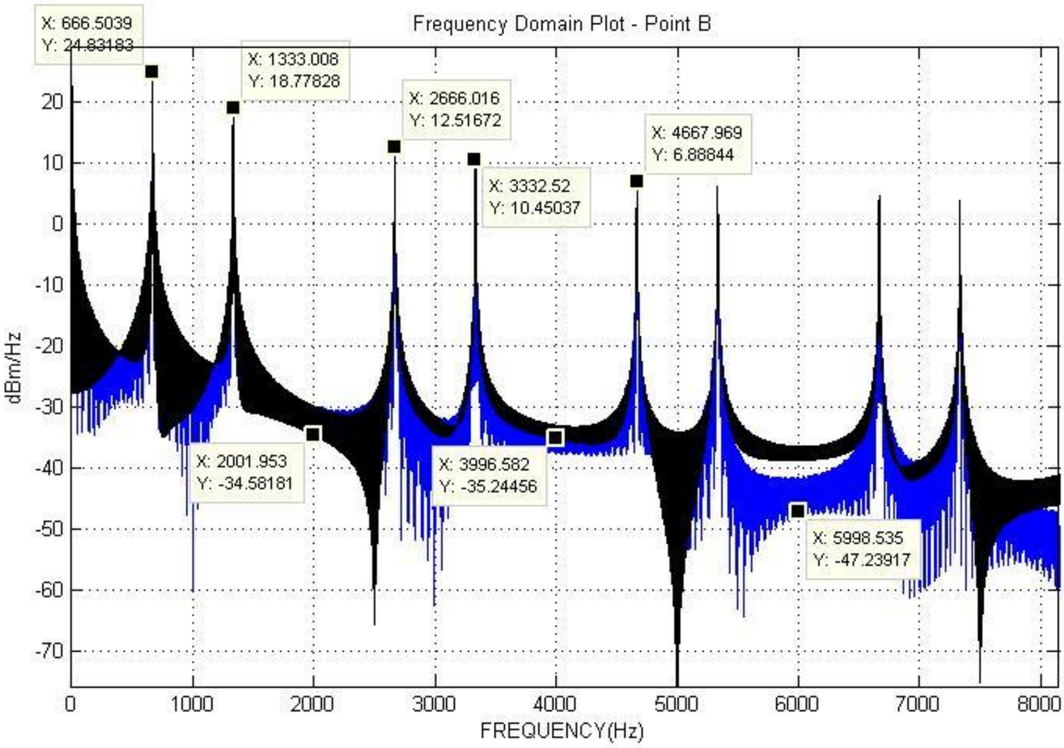


Figure 6.2.1 – Theoretical (black) and experimental (blue) unipolar NRZ PSD obtained by the programmed sequence 111110000000000_B

The generated time signal, $y(t)$, is composed by a sequence of elementary impulses with the same shape ($p(t)$) and amplitude given by $x(t)$ (delta Dirac impulses with amplitude a_k), as presented in equation (6.2.1). Knowing that $p(t)$ is a rectangular wave with width *C*, equation (6.2.2) is obtained.

$$x(t) = \sum_{k=-\infty}^{+\infty} a_k . \delta(t - k.t) \tag{6.2.1}$$

$$p(t) = rect\left(\frac{t}{C}\right) \tag{6.2.2}$$

, $y(t)$ is then translated into (6.2.3).

$$y(t) = p(t) * x(t) = rect\left(\frac{t}{C}\right) * \sum_{k=-\infty}^{+\infty} a_k . \delta(t - k.t) \tag{6.2.3}$$

In the frequency domain:

$$Y(f) = C.sinc(f.C) \times \frac{a_k}{T}.\sum_{n=-\infty}^{+\infty}.\delta\left(f - \frac{n}{T}\right) \tag{6.2.4}$$

In this experiment, the generated sequence with NRZ coding resembles to the transmission of rectangular pulses with period $T = 15.T_s$ and width equal to $C = 5.T_s$, where T_s is the period of each original symbol. Replacing C and T in equation (6.2.4), is obtained:

$$Y(f) = \frac{1}{3}.sinc(f.5.T_s) \times a_k.\sum_{n=-\infty}^{+\infty}.\delta\left(f - \frac{n}{15.T_s}\right) \tag{6.2.5}$$

The spectral power density is therefore given by:

$$\begin{aligned} S_y(f) = |Y(f)|^2 &= \frac{1}{9}.sinc^2(f.5.T_s) \times a_k^2.\sum_{n=-\infty}^{+\infty}.\delta\left(f - \frac{n}{15.T_s}\right) \\ &= \alpha \times \beta \end{aligned} \tag{6.2.6}$$

The picks of the PSD occur when $\beta = a_k^2$ and $\alpha \neq 0$:

$$f = \frac{n}{15.T_s}, \qquad n \in \mathbb{N}, \neq 3,6,9, \dots \tag{6.2.7}$$

, and the minimum values occur when $\beta = a_k^2$ and $\alpha = 0$:

$$f = \frac{n}{15.T_s}, \qquad n \in \mathbb{N}, n = 3,6,9, \dots \tag{6.2.8}$$

Is possible to observe that the obtained PSD depicted in figure 6.2.1 verifies the equations (6.2.7) and (6.2.8).

The DC component of a unipolar NRZ signal can be measured analytically through the area underneath the time domain graphic. As the number of words in the generated sequence increases, the DC component will take the value corresponding to the average of one pulse, as presented in equation (6.2.9)

$$\overline{x} = \frac{1}{15.T_s}\left(\int_0^{5.T_s} V^+\,dt + \int_{5.T_s}^{15.T_s} 0\,dt\right) \tag{6.2.9}$$

Since $V^+ = 2047$, the DC component of the unipolar NRZ signal is equal to approximately the level 682. The value obtained in the experiment mentioned before was 681,55, which is close to the expected value.

6.2.1.2 Random Sequence

In order to test a random generator with 16k samples, the sequence length must be set to at least $2^{12} - 1$, so that no symbol is repeated. In this experiment, was used a seed equal to five. The obtained PSD is presented in the figure 6.2.2.

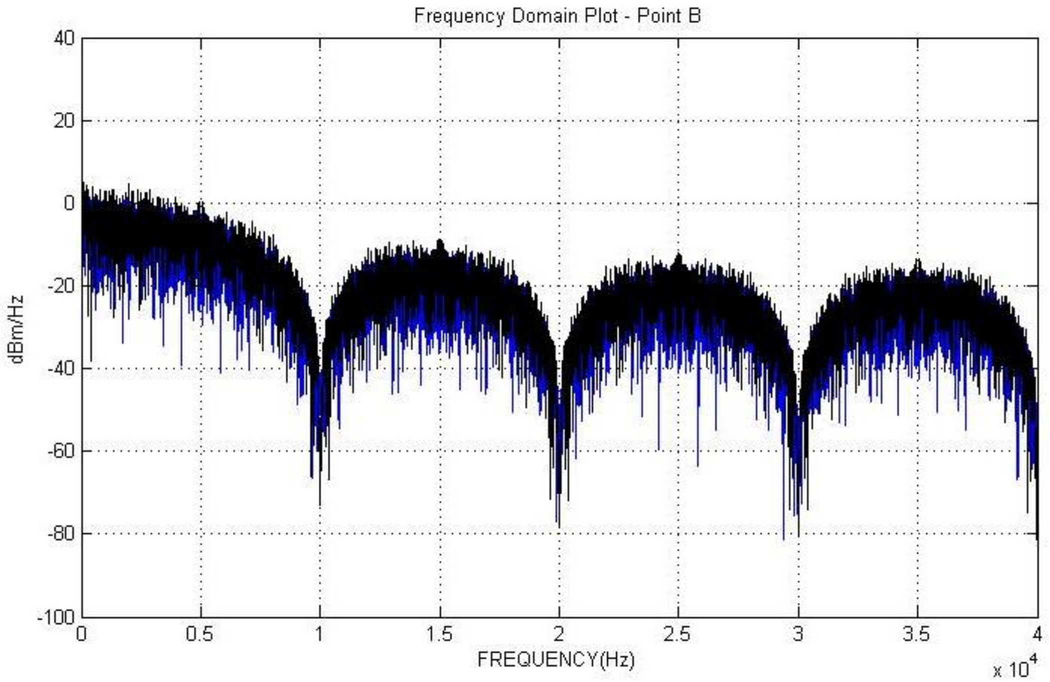


Figure 6.2.2 – Theoretical (black) and experimental (blue) unipolar NRZ PSD obtained by a random sequence

From [15], the PSD of a random delta Dirac impulse sequence is equal to (6.2.10), where R_n is the autocorrelation function (equal to (6.2.11)) and $E\{.\}$ is the sequence average.

$$S_x(f) = \frac{1}{T_s} \cdot \sum_{n=-\infty}^{+\infty} R_n e^{-jnw\,T_s} \tag{6.2.10}$$

$$R_n = E\{a_k \; a_{k+n}\} \tag{6.2.11}$$

Therefore, the PSD of the data signal sequence, $y(t)$, is equal to (6.2.12). For an on-off signaling, a_k can take a 0 or V^+ value, being $R_0 = E\{a_k^2\} = (V^+)^2 P(a_k = V^+) + (0)^2 P(a_k = 0) = \frac{(V^+)^2}{2}$ and $R_n = E\{a_k\ a_{k+n}\} = \frac{(V^+)^2}{4}$, for equiprobable a_k levels.

$$S_y(f) = |P(f)|^2 S_x(f) = \frac{|P(f)|^2}{T_s} . \sum_{n=-\infty}^{+\infty} R_n e^{-jnw\, T_s} \tag{6.2.12}$$

Replacing R_0 and R_n into the equation (6.2.12), and with some mathematical manipulation the data signal sequence PSD is obtained as shown in equation (6.2.13).

$$S_y(f) = \frac{|P(f)|^2}{4.T_s} . (V^+)^2 . \left[1 + \frac{1}{T_s} \sum_{n=-\infty}^{+\infty} \delta\left(f - \frac{n}{T_s}\right)\right] \tag{6.2.13}$$

,where $\frac{|P(f)|^2 . (V^+)^2}{4.T_s}$ defines the PSD outer shape and $\frac{|P(f)|^2 . (V^+)^2}{4.T_s^2} . \sum_{n=-\infty}^{+\infty} \delta\left(f - \frac{n}{T_s}\right)$ defines their peaks.

For the NRZ unipolar signal, $P(f) = T_s . sinc(f . T_s)$ and $\frac{|P(f)|^2 . (V^+)^2}{4.T_s} = \frac{(V^+)^2 . T_s . sinc^2(f . T_s)}{4}$. Therefore, (6.2.13) results in a zero value when $f = \frac{n}{T_s}, n \in \mathbb{N}$, which corresponds to multiples of the symbol rate, as seen in figure 3.2.1a. The peak value is obtained only at the null frequency, since is the only moment when the functions *sinc*(.) and Dirac have simultaneously an non-null value.

The experimental result shown in figure 6.2.2 verifies the conditions mentioned above: have only one discrete component in the DC component and the minimum values are presented at integer multiples of the symbol rate ($R_s = 10\text{kHz}$).

The relative maximum values that appear in the experimental spectrum are given by the *sinc* function maximum values, which occur when:

$$\begin{cases} f = \dfrac{2n+1}{2T_s}\ Hz\ , n \,\epsilon\, \mathbb{N} \\ \quad f = 0\ Hz \end{cases} \tag{6.2.14}$$

In accordance with (6.2.14) and the current experiment, the first, second and third maximum values appear at $f = 0$, $f = 15kHz$ and $f = 25kHz$, as shown in figure 6.2.2.

The RZ unipolar elementary pulse, $p(t)$, has half the width of the unipolar NRZ elementary pulse, therefore: $C = \frac{T_S}{2}$.

$$p(t) = rect\left(\frac{t}{\frac{T_S}{2}}\right) \overset{F}{\Leftrightarrow} P(f) = \frac{T_S}{2} sinc\left(f.\frac{T_S}{2}\right) \tag{6.2.15}$$

Substituting (6.2.15) into (6.2.13), results in:

$$S_y(f) = \frac{T_s}{16}.(V^+)^2.sinc^2\left(f.\frac{T_s}{2}\right).\left[1 + \frac{1}{T_s}\sum_{n=-\infty}^{+\infty} \delta\left(f - \frac{n}{T_s}\right)\right] \tag{6.2.16}$$

As previously seen, the first parcel of (6.3.16), that contains only the *sinc*(.) function, defines the envelope of the PSD. Therefore, the PSD null values occur when $f = \frac{2.n}{T_s}, n \in \mathbb{N}$. In figure 6.2.3 is presented the PSD of a random sequence with a unipolar RZ coding, where it is possible to observe the first minimum values at $f = 20KHz$ and $f = 40KHz$.

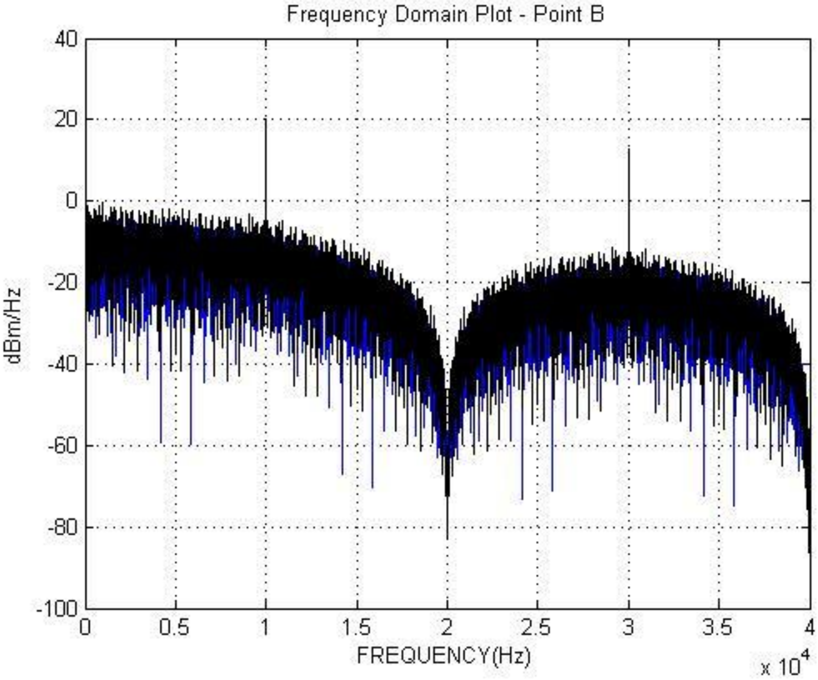
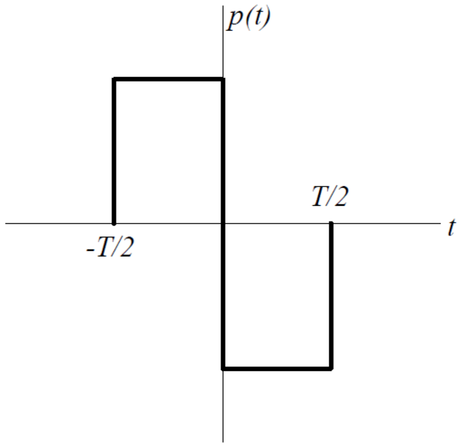


Figure 6.2.3 – Theoretical (black) and experimental (blue) unipolar RZ PSD obtained by a random sequence

The second component defines the spectrum lines, which occur when the *sinc* function and the delta Dirac functions are different from zero simultaneously, namely, when $f = \frac{n}{T_s}, n\ odd\ \wedge\ n = 0$. Once more the experimental results are in accordance with the theoretical expectations: the first three discrete lines occur at $f = 0Hz$, $f = 10kHz$ and $f = 30kHz$. The relative maximum values are more restrict than the peaks of the PSD, happening when $f = \frac{2.n+1}{T_s}, n \in \mathbb{N}$.

In the case of the Manchester coding, the area of the elementary pulse is zero, which leads to the form depicted in figure 6.2.4 and the equation 6.2.17. An average of 0,61 was obtained for an experiment with 16k samples, that approximates the area of the Manchester elementary pulse.



$$p(t) = rect\left(\frac{t+\frac{T_S}{4}}{\frac{T_S}{2}}\right) - rect\left(\frac{t-\frac{T_S}{4}}{\frac{T_S}{2}}\right)$$

$$\overset{F}{\Leftrightarrow}$$

$$P(f) = j\,T_S.sinc\left(f.\frac{T_S}{2}\right).sin\left(\frac{w.T_S}{4}\right)$$

Figure 6.2.4 – Elementary Manchester Pulse (6.2.17)

The highest level and the lowest level take now all the 12 bit range, with $V^{+} = 2047$ and $V^{-} = -2047$, respectively. As a result, the autocorrelation take the values: $R_0 = E\{a_k^2\} = (V^{+})^2 P(a_k = V^{+}) + (V^{-})^2 P(a_k = V^{-}) = |V|^2$ and $R_n = E\{a_k\ a_{k+n}\} = 0$. Replacing these autocorrelation values and the pulse shape form (6.2.17) into equation (6.2.12), results into the sequence PSD in (6.2.18)

$$S_y(f) = \frac{|P(f)|^2}{T_s}.\sum_{n=-\infty}^{+\infty} R_n e^{-jnw\,T_s} = \frac{|P(f)|^2}{T_s}.|V|$$
$$= |V|^2.T_s.sinc^2\left(f.\frac{T_s}{2}\right).\sin^2\left(\frac{\pi.f.T_S}{2}\right) \qquad (6.2.18)$$

Through the mathematical manipulation of (6.2.18), is possible to determine the frequencies at which the null values are localized, as shown in equation (6.2.19).

$$S_y(f) = 0 \implies sinc\left(f.\frac{T_s}{2}\right) = 0 \vee \sin\left(\frac{\pi.f.T_S}{2}\right) = 0 \implies f = \frac{2n}{T_S}, n \in \mathbb{N}_0 \qquad (6.2.19)$$

$S_y(f)$ will take the maximum value when the sine function is maximum, as shows equation (6.2.20).

$$\sin\left(\frac{\pi.f.T_S}{2}\right) = 1 \Longrightarrow \frac{\pi.f.T_S}{2} = \frac{\pi}{2}(2n+1) \;\Rightarrow f = \frac{(2n+1)}{T_S}, n \in \mathbb{N}_0 \qquad (6.2.20)$$

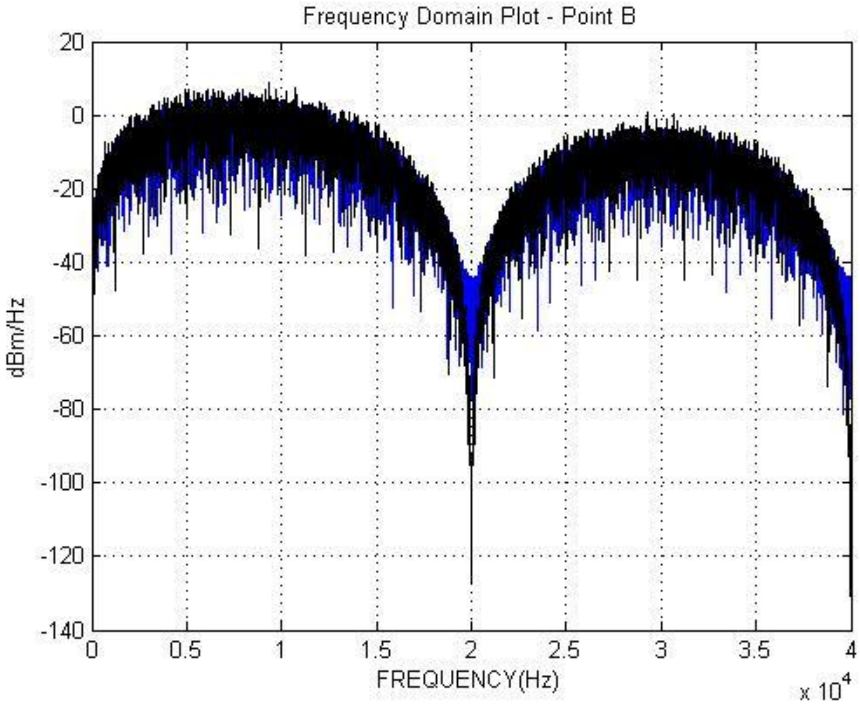
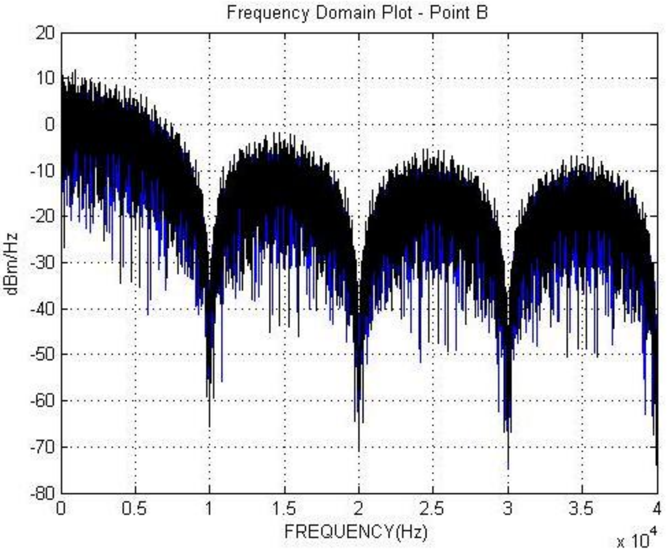


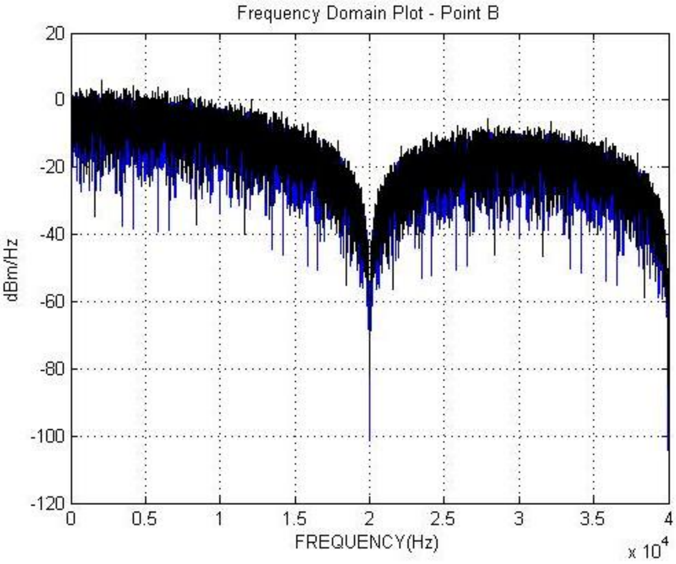
Figure 6.2.5 – Theoretical (black) and experimental (blue) Manchester PSD obtained by a random sequence

The experimental PSD presented in figure 6.2.5, corroborates the expected minimum values ($S_y(f) = 0$) for 0, 20 and 40 kHz (separation of 20 kHz) and the relative maximums at 10 and 30 KHz (also with a separation of 20 kHz). Since no delta Dirac function exists in the Manchester PSD, no peaks are observed in it.

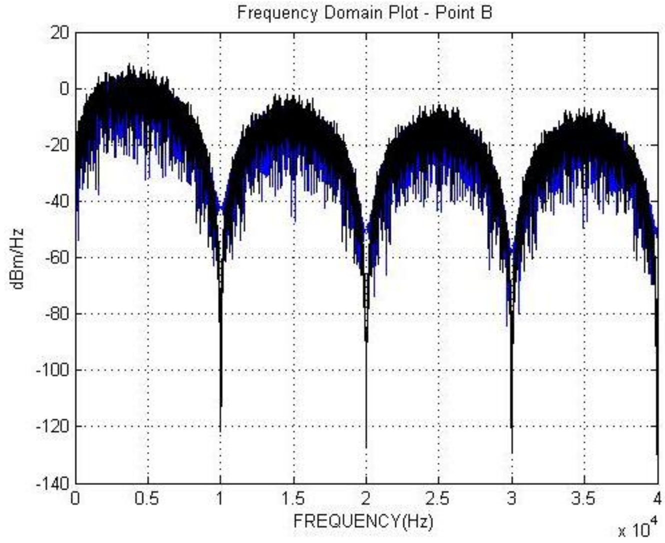
The following plots in figure 6.2.6 presents the PSD of the remaining signals that the system can also generate. By their careful analysis and comparing with the plots in figure 3.2.1 and the theoretical graphs, is possible to infer that the line coding block processes correctly the data generated. Is important to mention that the graphics presented in figure 3.2.1 have linear amplitudes in the y axis, whereas the plots presented through simulation have a decibel scale.



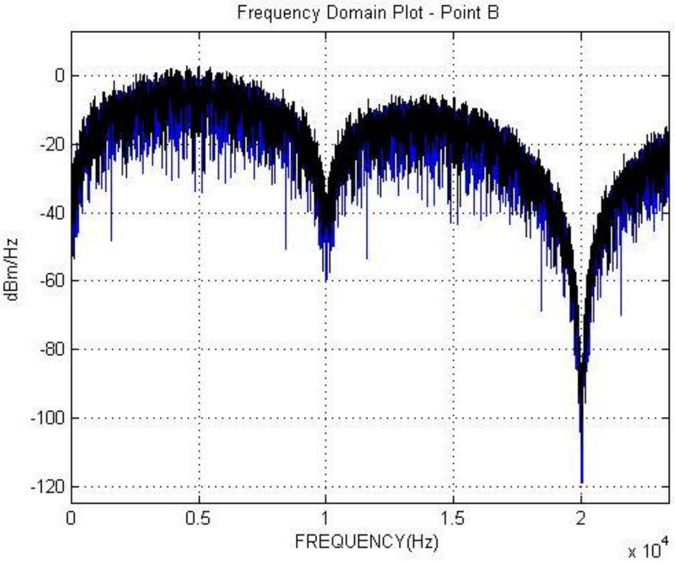
a) Polar NRZ



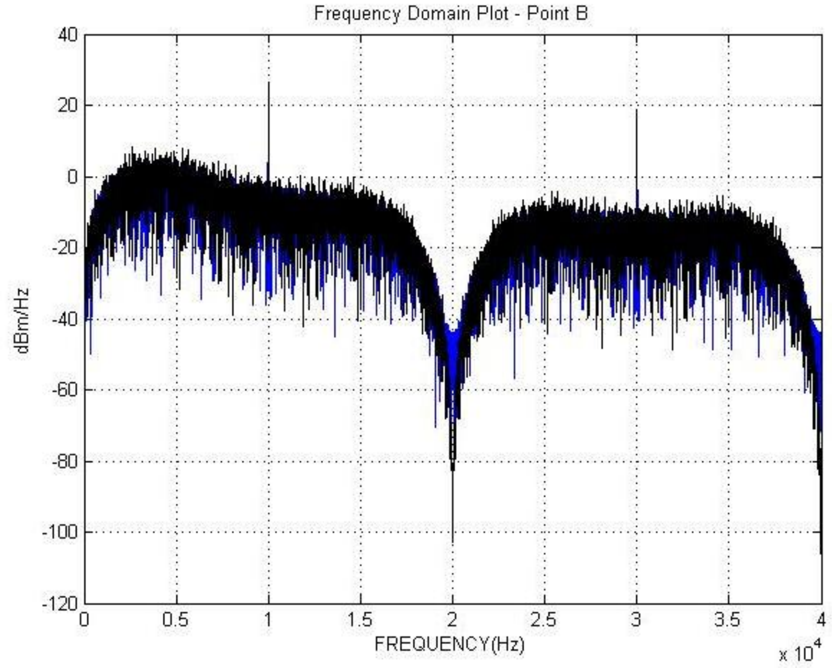
b) Polar RZ



c) Bipolar NRZ
(AMI NRZ)



d) Bipolar RZ
(AMI RZ)



e) CMI

Figure 6.2.6 – Theoretical (black) and experimental (blue) PSDs for different line codes

The theoretical graphs presented in this section were obtained through the Matlab, by pseudorandom values drawn from the standard uniform distribution on the open interval (0,1).

6.2.2 Raised Cosine

After the signal has been processed by the line coding block, it can go through the raised cosine block. This block will only allow non-return-to-zero signals at its input. This condition will enable interpolation emulation, by letting only one sample of each symbol be different than zero, and the rest of the samples are set to zero. An interpolated signal example for a sampling factor of 8, is presented in figure 6.2.7a, being a result of an unipolar NRZ coding of a signal reduced by four in amplitude. That is, the original unipolar NRZ signal with amplitude of $V^+ = 2047$ is first reduced to $V^+ = 511$ and next interpolated. As expected, due to ISI, the resultant filtered signal will increase the impulses amplitude when several bits at level ‘1’ are transmitted (fig. 6.2.7b). This maximum amplitude increase will lead to a filter response higher than 0 dB. If the 23 filter taps aren’t divided by the sample factor upon its creation and there are 8 samples/symbol, the maximum filter response amplitude is equal to $20.\log(8) = 18{,}06dB$. However, for the example mentioned in figure 6.2.8b the filter response taps were divided by the sampling factor upon the graph creation, which has resulted into a maximum 0 dBm for 0 Hz.

The data signal was normalized by $V^{+} = 2047$ and has a maximum equal to 511, therefore, the maximum PDS will be $10log10(4) = 6{,}02$ dB lower than zero. Despite the filter taps were divided by the sampling factor upon the graphics representation, the 18,06 dB is still present when the signal 'D' and 'E' are compared at 0 Hz.

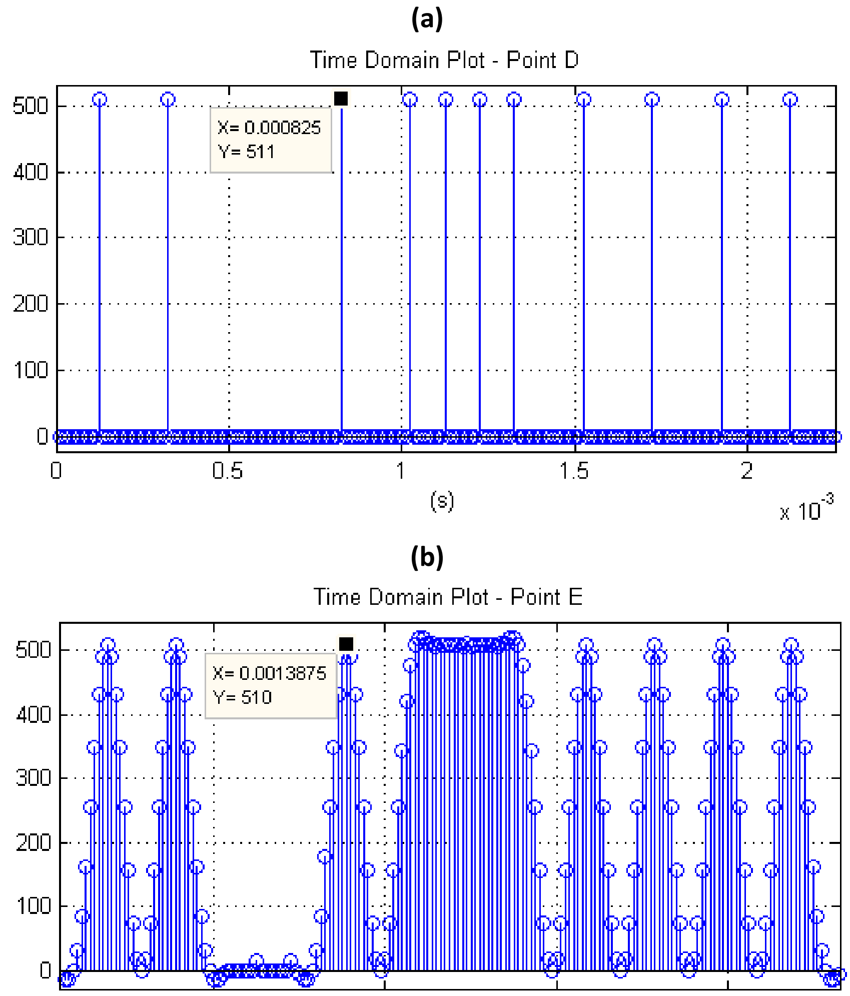
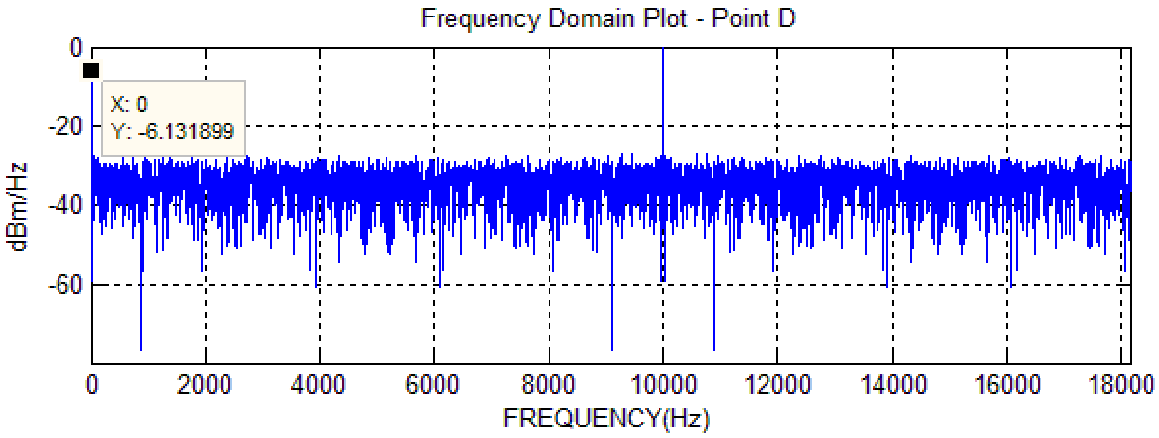


Figure 6.2.7 – Time domain unipolar NRZ signal after a) interpolation and b) after the raised cosine filter for r=1.

(a)



(b)

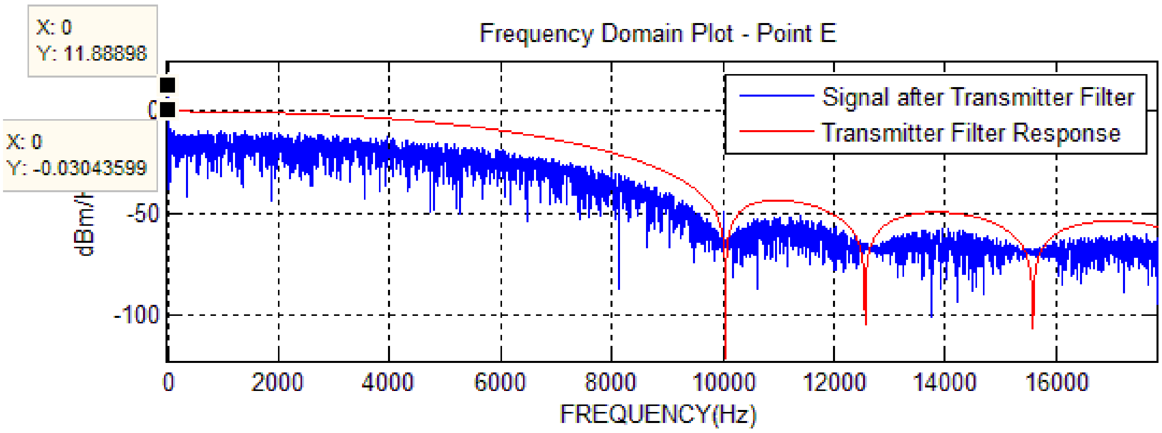
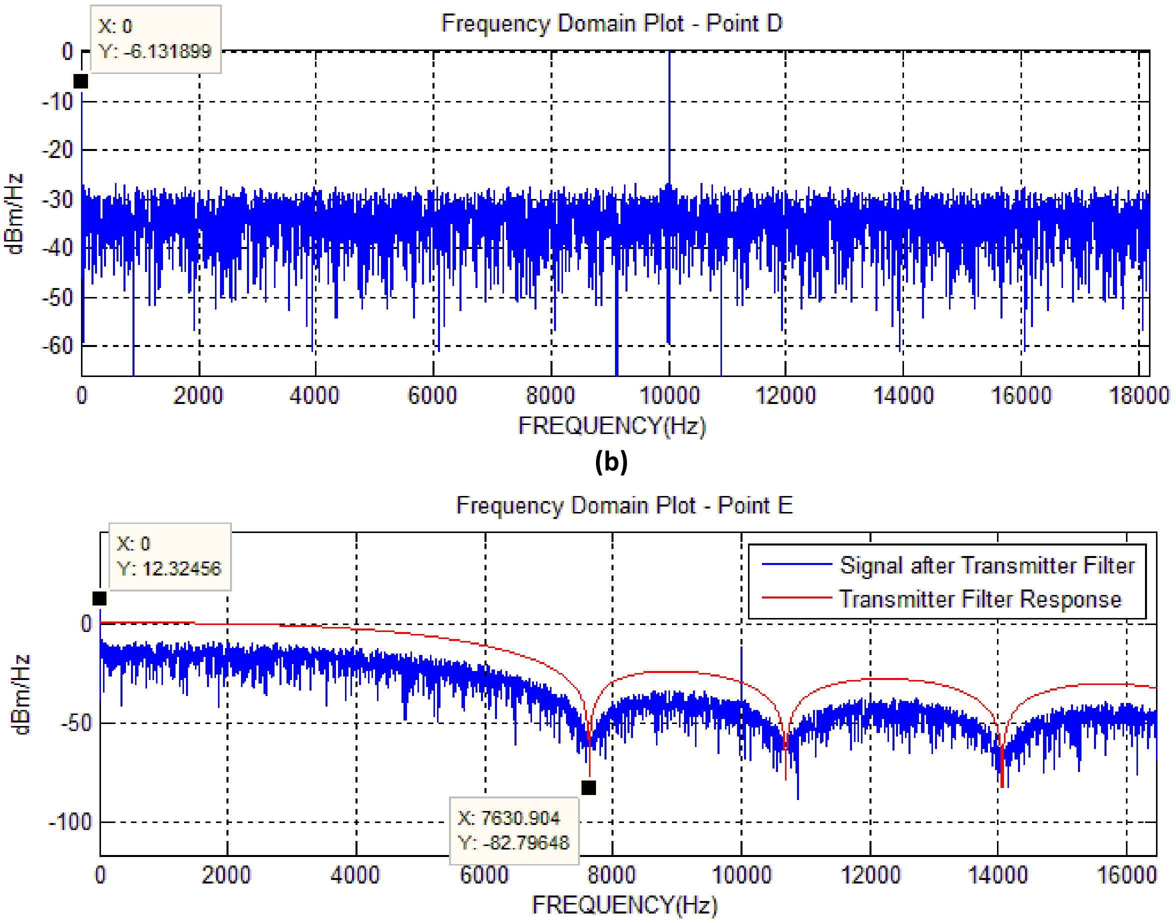


Figure 6.2.8 – a) Interpolated signal PSD before and b) after the raised cosine filter (Blue) and the filter response (Red) for r=100%

The previous experiment was done with a roll-off of 100% (r=1), which means that the bandwidth of the first lobe is delimited by $B_T = \frac{(1+r).R_s}{2} = R_s = 10KHz$, as shown in figure 6.2.8b. On the other hand, the 50% roll-off, shown in figure 6.2.9b as smaller bandwidth, given by approximately: $B_T = \frac{(1+r).R_s}{2} = 0{,}75.R_s = 7{,}5KHz$. Since the bandwidth is smaller in the last case, the pulse width will be wider and will decay less rapidly with more signal ringing. These oscillations will lead to a larger ISI outside the sampling instants, resulting in pulse amplitude increase when several "1" bits are sent together, as represented in figure 6.2.10b.

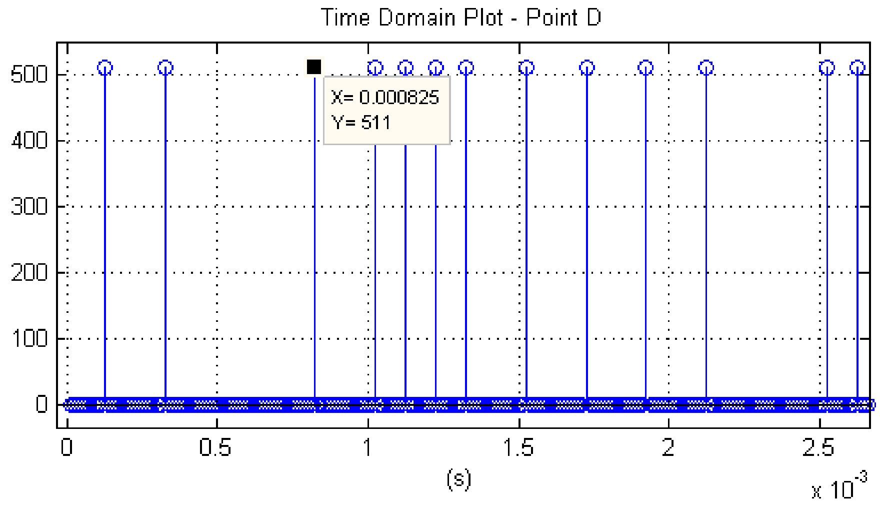
(a)



(b)

Figure 6.2.9 – a) Interpolated signal PSD before and b) after the raised cosine filter (Blue) and the filter response (Red) for r=50%

In either case, the delay between the interpolated input and the filtered output maintains equal to 0,56 ms and the instants in which the time signal passes through zero.



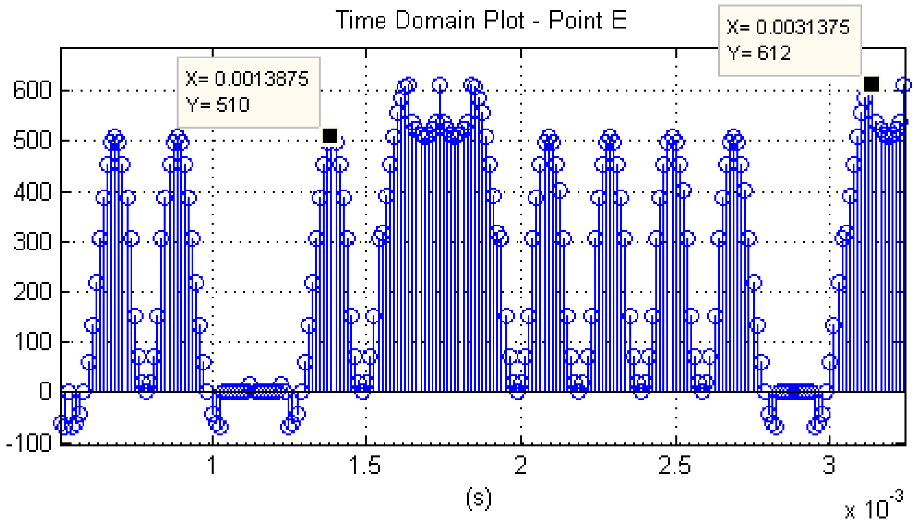


Figure 6.2.10 – Time domain unipolar NRZ signal after a) interpolation and b) after the raised cosine filter for r = 0.5

Another way of analyzing the signal characteristics is through their eye diagram. In Figure 6.2.11 is depicted the eye diagram plot for both the experimented signals. Is possible to see the ISI influence on both eye plots. Despite the jitter being approximately the same, the noise margin is lower in the signal with 50% roll-off factor, which results in a higher ISI outside the sampling instants, than the case with a roll-off equal to 100%. As expected, the optimum decision points are located at half the symbol period in both experiments.

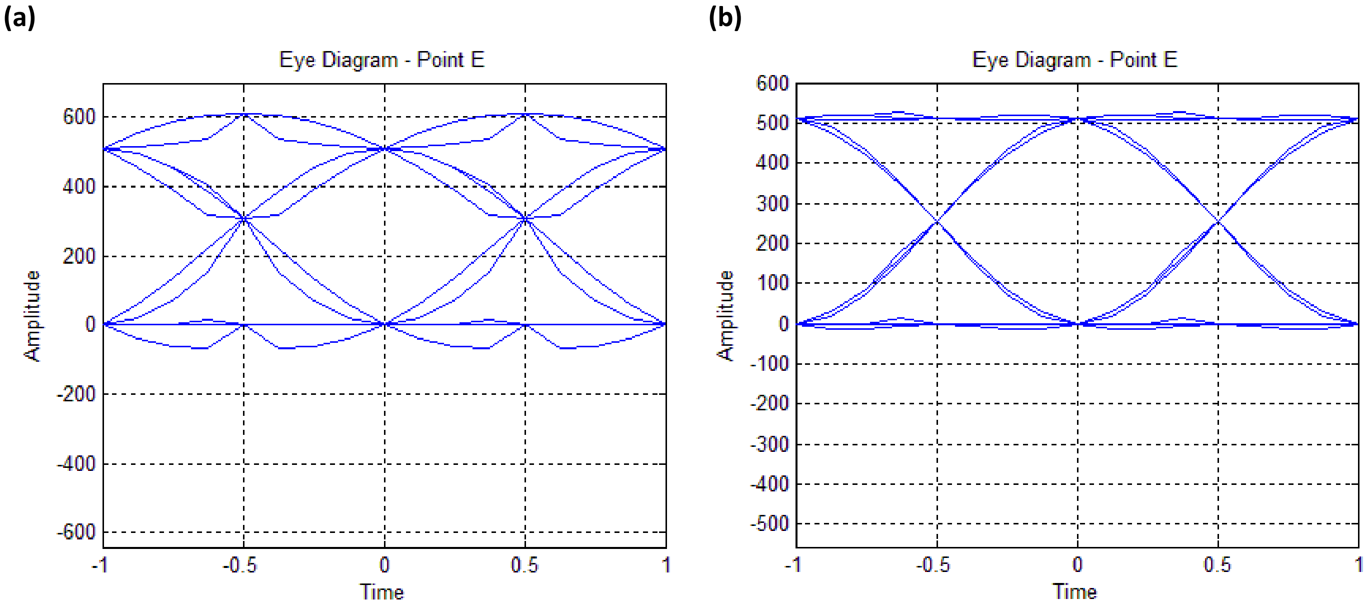


Figure 6.2.11 – Eye diagram for the signal after the raised cosine filter with a) r=0.5 and b) r=1

The plots presented here, were obtained by analyzing a signal with 16000 samples, due to the system restrictions when two signals are plotted simultaneous in the same

FPGA. In order to decrease the computation time, only 256 symbols are used for representing the eye diagram. Is important to mention that the eye diagram can only be drawn if the data signal has more than 256 symbols, since this is a predefined value in the system.

If we run the same test, but know with just 4 samples/symbol (fewer than before), is possible to observe that the jitter is higher if the data signal is filtered by a 50% raised cosine filter. This effect is known as telegraphic distortion and is depicted in the eye diagrams of figures 6.2.12a and 6.2.12b.

The FIR filter that runs in the FPGA, work with 23 taps, which are taken from the raised cosine infinite impulse response. Considering the case where the user sets the sampling factor to 32, the 23 filter taps will not allow any ISI or jitter, since the last 9 samples of each symbol are zero after it has been filtered. If the number of samples/symbol is lower than 23, the ISI occurs. The jitter will only appear when there is 4 samples/symbol since the number of overlaps into the same symbol is large enough to allow this effect. This effect should be observed also in the 8 and 16 samples/symbol cases, however, due to the FPGA filter design, is not possible to achieve such results.

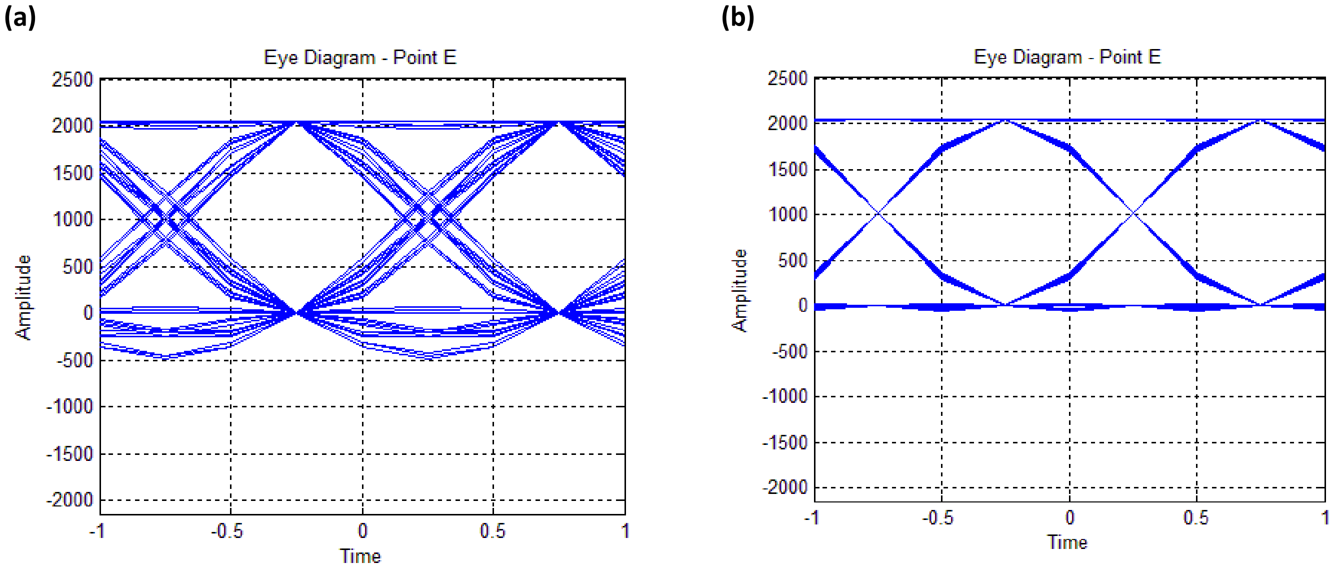


Figure 6.2.12 – Eye diagram for the signal (with 4 samples/symbol) after the raised cosine filter with a) r=0.5 and b) r=1

6.3 *Low Pass Filter*

The laboratorial module provides the alternative of using a pulse shaping filter based on a low-pass Butterworth filter instead of using a raised cosine filter. The Butterworth filter was implemented using a Matlab pre-conceived function which restricts the filter cut-off frequency between $[0{,}113 \times F_s/2, 0{,}789 \times F_s/2]$ Hz, where $F_s = R_s.Sampling\ Factor$. The signal that enters the low-pass Butterworth filter is the original (none interpolated) line coding output, as shown in figure 6.3.1a example.

Once more, the experiment will be conducted with 16000 samples of a unipolar NRZ coded signal. The filter cut-off frequency was set to 7,5kHz, as seen in figure 6.3.2b. At the 3 dB filter response, the experiment shows that the difference between the original signal and the filtered signal was equal to 2,94 dB. This discrepancy occurs due to the truncation into three decimal digits of the filter coefficients before going to the FPGA FIR filter and due to the filter order.

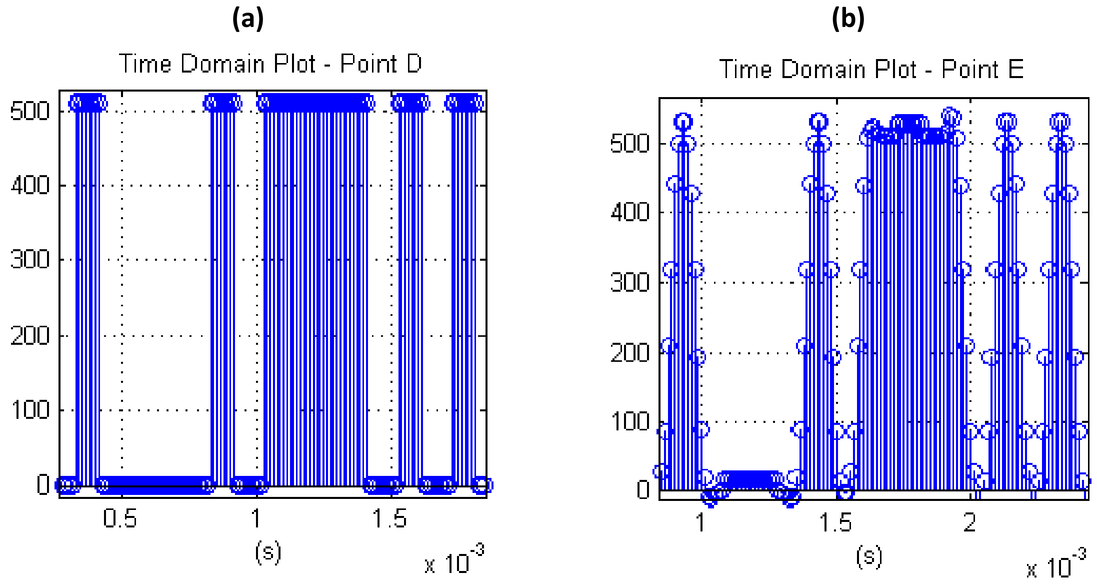
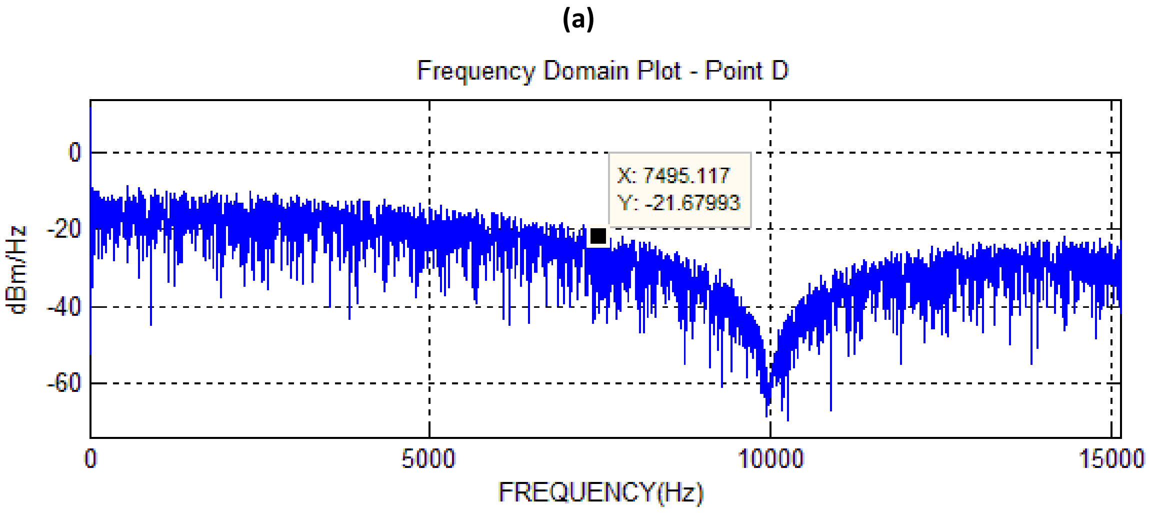


Figure 6.3.1 – Time domain unipolar NRZ signal a) before and b) after the low-pass Butterworth filter



(b)

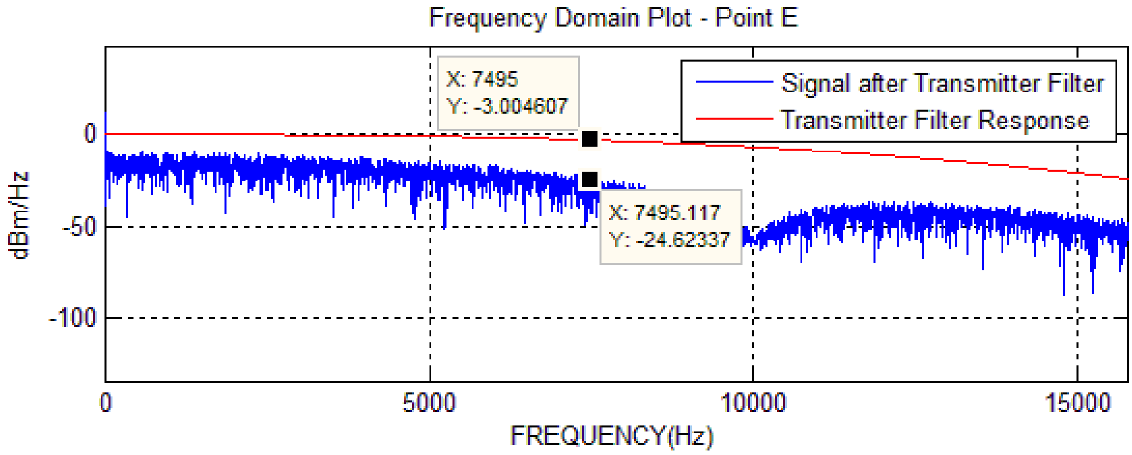


Figure 6.3.2 – Unipolar NRZ PSD a) before and b) after the low-pass Butterworth filter

6.4 Channel Filter

Like in the case of the previous blocks, in order to test the designed channel filter block, the samples will be generated at a symbol (bit) rate of $R_s = 10K\ symbols/s$ and a sampling factor of $8\ samples/symbol$.

6.4.1 High-Pass

The channel can be emulated using one of two filters: a high pass filter and a filter that resembles the effect of a coaxial cable. The high-pass filter is implemented with a linear-phase normalized FIR filter that has a magnitude response equal to 0 dB at the center frequency of the pass-band.

To this experiment, the 16000 generated samples were formatted with a unipolar NRZ code, being the direct input of the channel filter. This way, only the channel filter is tested. The data signal have a maximum of $V^+ = 511$ which is then normalized by $V^+ = 2047$, as in previous experiments upon the graphic creation .The 6dB cut-off frequency was set to 7,5kHz, as seen in figure 6.4.1b. As expected, at the stop-band, the filter response decreases and the data signal becomes attenuated. The phase in the pass-band has a linear decay as depicted in figure 6.4.2.

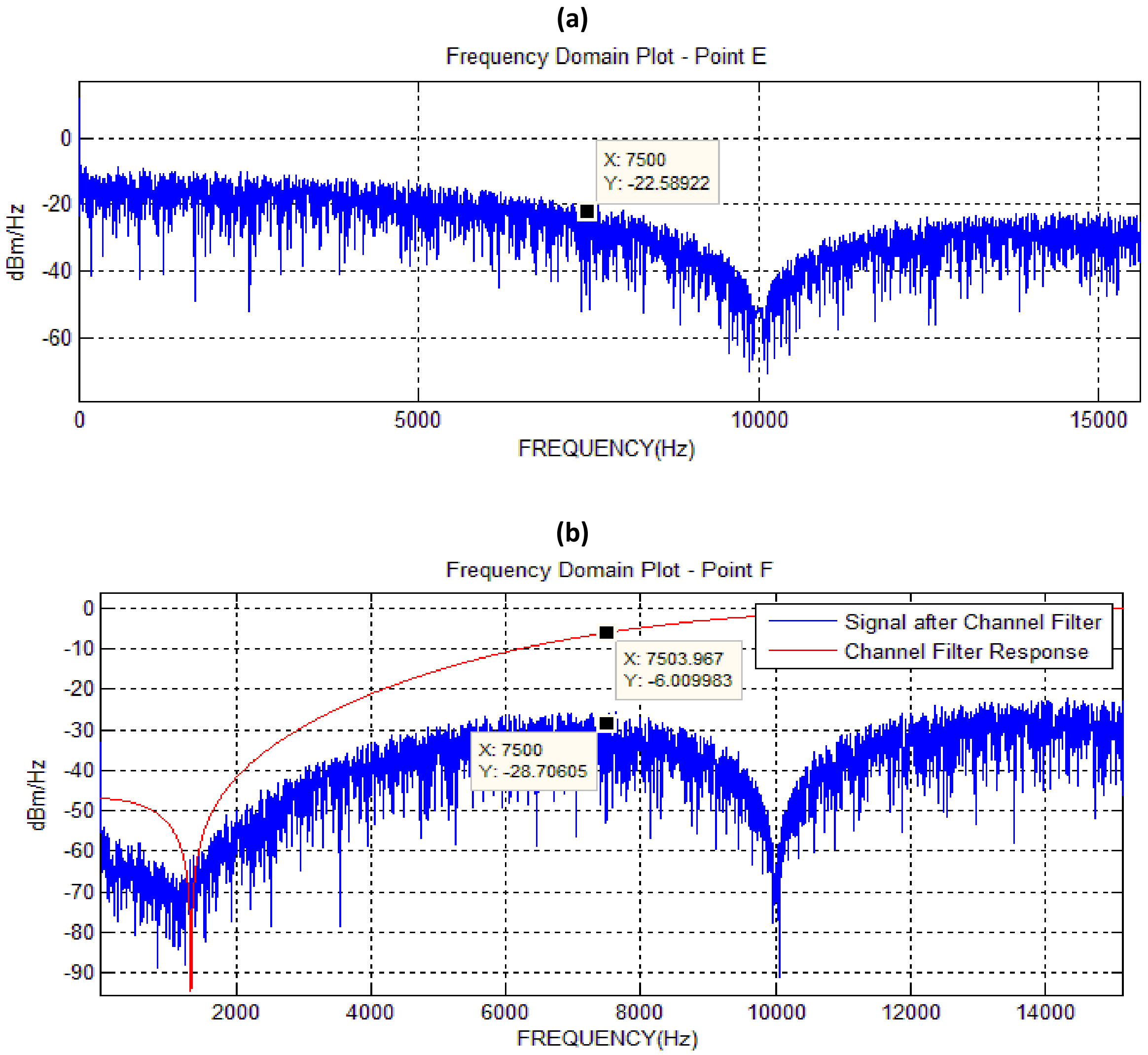


Figure 6.4.1 – Unipolar NRZ PSD a) before and b) after the channel high-pass filter

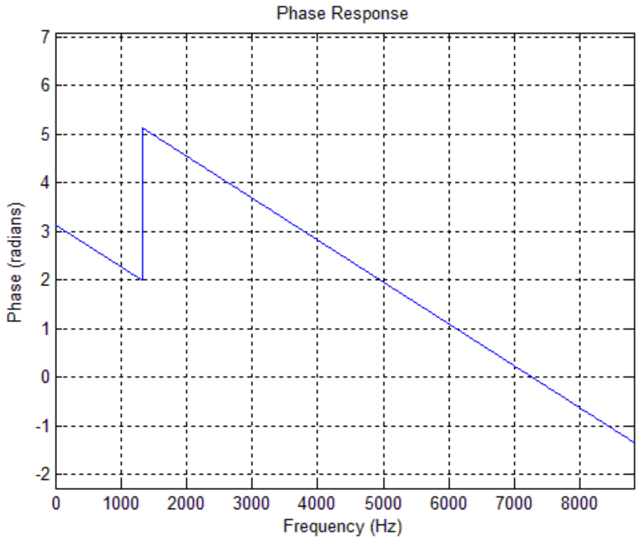


Figure 6.4.2 – High-Pass Filter Phase

6.4.2 Coaxial Cable

The parameters used to obtain the coaxial filter impulse response were taken from [16], a semi-rigid coaxial cable produced by Micro-Coax, which are described in the table 6.4.1.

Parameter	Value
Inner conductor diameter	$0{,}511.10^{-3}\ m$
Outer conductor diameter	$3{,}581.10^{-3}\ m$
Relative dielectric permittivity	$2.1\ (PTFE/Teflon)$
Relative dielectric permeability	1
Inner conductor conductivity	$5{,}8.10^{7}\ S/m\ (Copper)$
Outer conductor conductivity	$5{,}8.10^{7}\ S/m\ (Copper)$

Table 6.4.1 – Coaxial Cable Parameters

The user can only define the cable length in the Matlab analyzer. In figure 6.4.3 is presented the filter response at -3dB for a cable with length ranging from 200 to 2000 meters, working at the sampling frequency $F_s = 80kHz$. It was taken 200 points to build that plot.

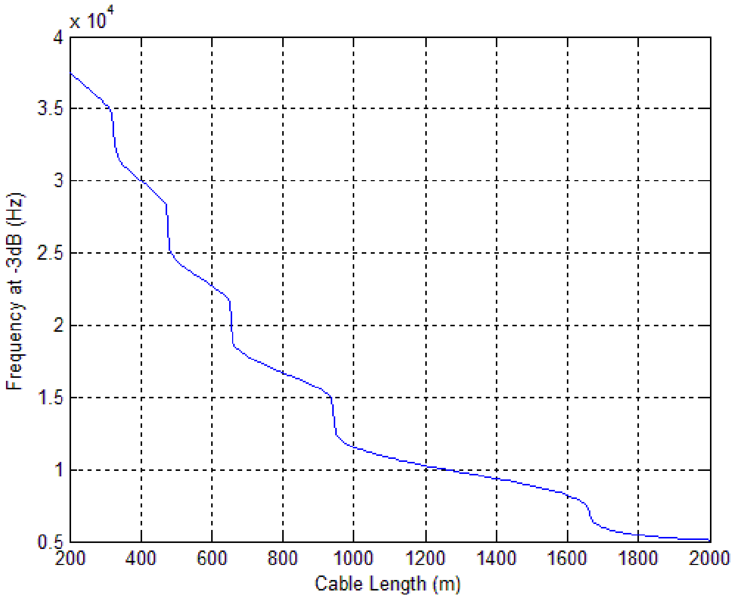


Figure 6.4.3 – Coaxial cable filter response at -3dB for several cable lengths

6.4.3 AWGN

The experiments conducted in this section will focus only on the noise generator itself. In figure 6.4.4a and 6.4.4b is presented the logarithmic histogram with 100 class intervals for two situations: a standard deviation of 0.5 and 1, respectively. The bilateral noise PSD (in dBm/Hz) used in each experiment is given in equation (6.4.1).

$$\sigma = \sqrt{\frac{\eta}{2}.F_s} \Longrightarrow \begin{cases} \sigma = 0{,}5 \Longrightarrow \frac{\eta}{2} = \frac{(0{,}5)^2}{80.10^3} = 3{,}125.10^{-6}\ (w/Hz) = -25\ (dBm/Hz) \\ \sigma = 1 \Longrightarrow \frac{\eta}{2} = \frac{1}{80.10^3} = 3{,}125.10^{-6}\ (w/Hz) = -19\ (dBm/Hz) \end{cases} \qquad (6.4.1)$$

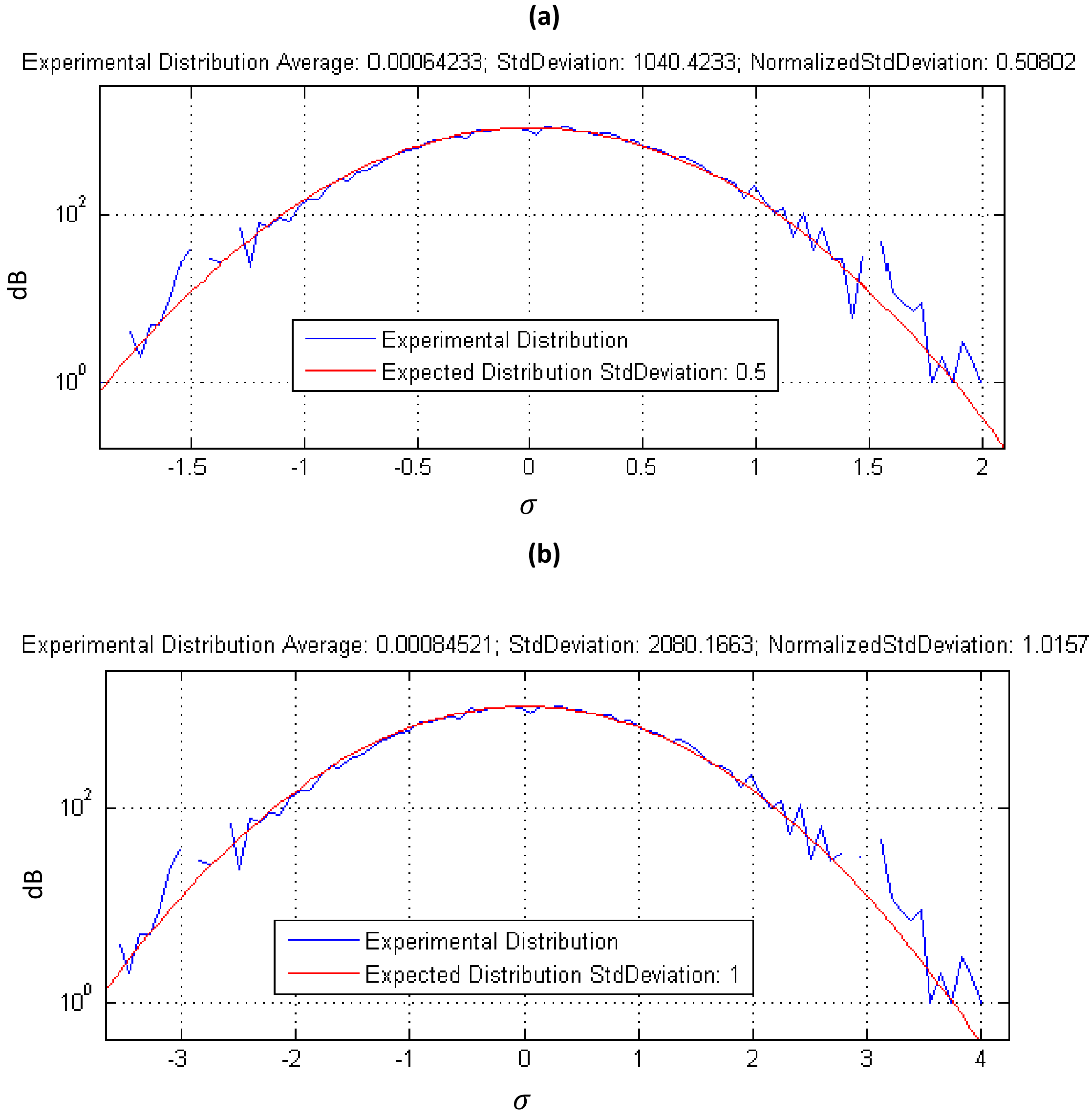


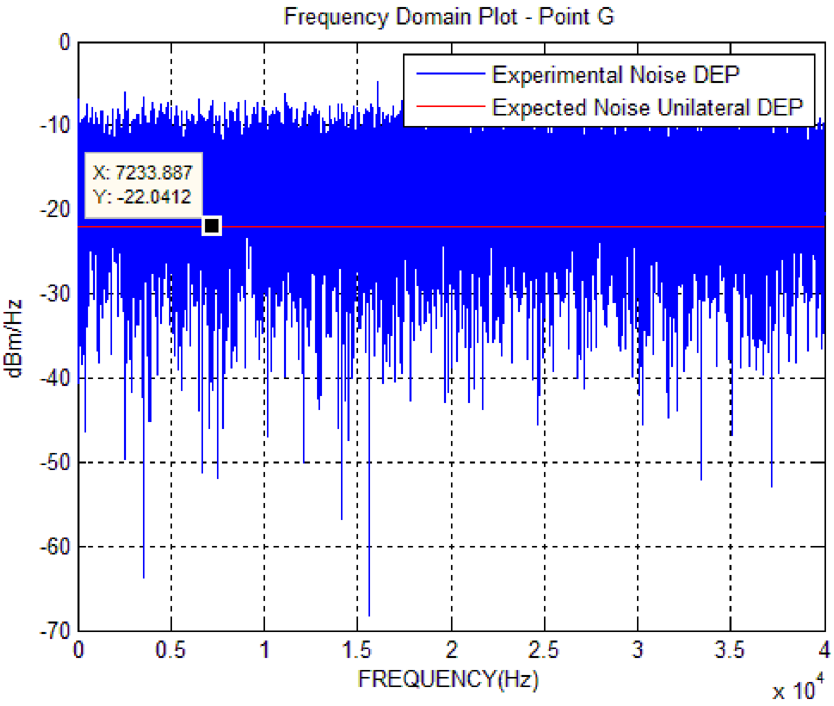
Figure 6.4.4 – Noise generator logarithmic histogram for a) $\sigma = 0{,}5$ and b) $\sigma = 1$

As expected, in both cases, the generated samples have a Gaussian distribution, centered in approximately zero value. In each experiment were analyzed 32000 samples and as the number of generated samples increases, they tend to a zero average. The standard deviation stands for the deviation in order to the highest level in the system: 2047. Therefore, to a normalized standard deviation of 0.5, the noise samples set must have a standard deviation approximately equal to 1024.

In figure 6.4.5 is presented the comparison between the expected and the experimental unilateral PSD for both experiments. The unilateral noise power is obtained by doubling the bilateral noise power. In the decibel scale it corresponds to add 3 dB to the bilateral DEP, which results into (6.4.2).

$$\begin{cases} \sigma = 0{,}5 \Longrightarrow \eta = -22\ (dBm/Hz) \\ \sigma = 1 \Longrightarrow \eta = -16\ (dBm/Hz) \end{cases} \tag{6.4.2}$$

(a)



(b)

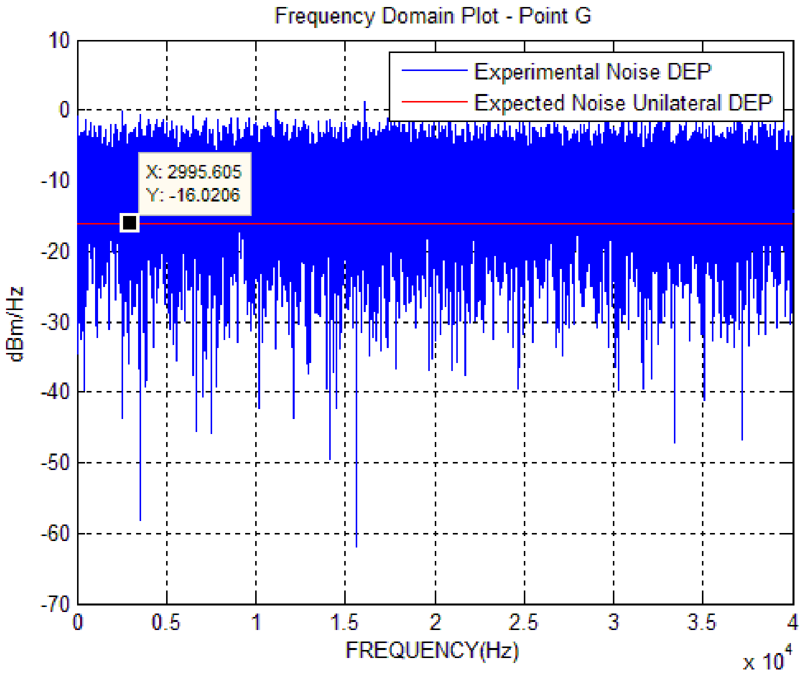


Figure 6.4.5 – Noise generator PSD for a) $\sigma = 0{,}5$ and b) $\sigma = 1$

Since the noise that is generated is white, the PSD in both cases will have an approximately uniform distribution for the given frequency range.

Despite only two cases were presented, the user can choose between the bilateral PSDs that verify the condition: $\sigma \in [0{,}1]$.

6.5 Peripherals

6.5.1 DAC

As mentioned in section 4.3.1, the chosen DAC can operate ideally until 1.47 MHz. In order to know what the real operation rate is, a random sequence with a bit rate of 1MHz and a sequence length of $2^{12} - 1$ was generated. This stream was then formatted with the polar NRZ line coding (presented in figure 6.5.1 as the blue signal). In the experiments done so far, the signal was generated through the *Step-by-Step* mode, since only a given set of samples needs to be analysed. However, since the DAC response needs to be analysed in real time using an oscilloscope, the *Continuous* mode was used. The DAC output is shown in figure 6.5.1 red plot. In order to achieve a better accuracy when comparing these two signals, instead of presenting them in an oscilloscope, they were sampled through the *Tektronix OpenChoice Desktop* software with 1024 samples. The original (blue) signal was obtained through a dedicated digital output in the FPGA, which was connected to the oscilloscope channel 1.

In order to determine the DAC response, the gathered data was sent to Matlab to perform the difference between the two PSDs (blue and red plots). However, since this difference has a lot of fluctuations, an average with consecutive 5 samples, was determined. From this average, the black plot in figure 6.5.1 was obtained. It is important to mention that this average will lead to a delay of 5 samples, therefore, the actual 3dB frequency occurs at 322 kHz, instead at the 334 kHz. This experimental result shows that the DAC frequency operation limit is less than the theoretical value (1,47MHz).

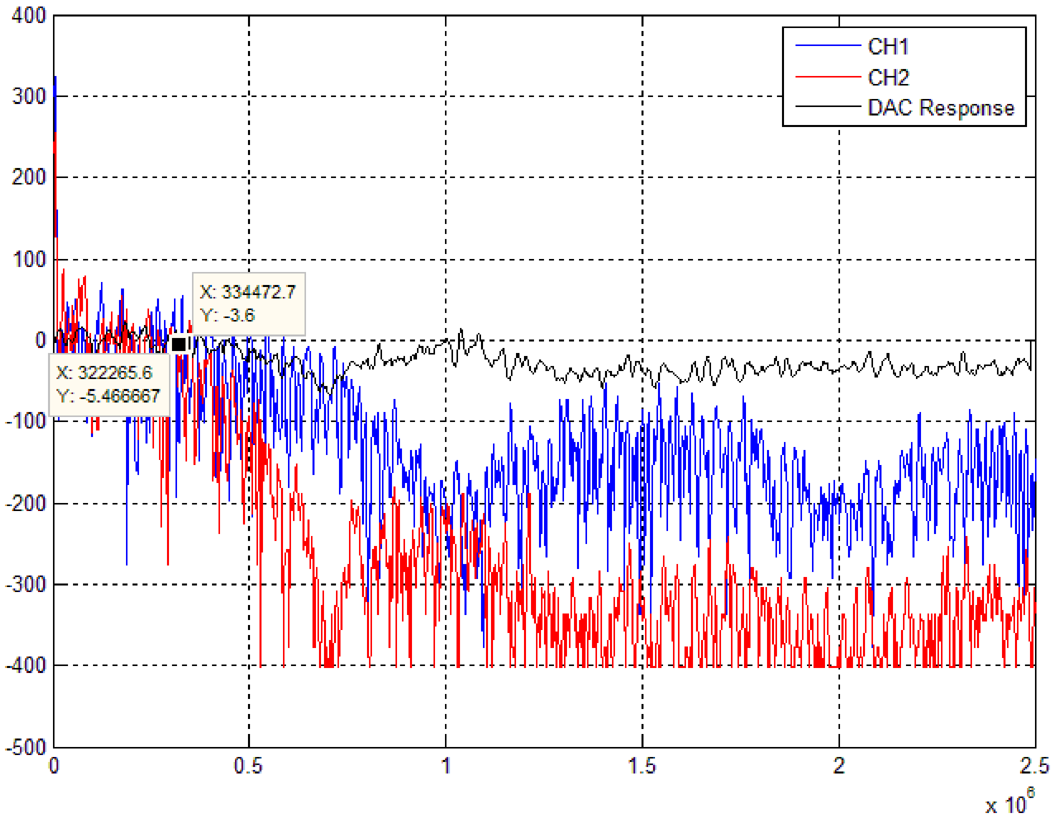


Figure 6.5.1 – DAC input (CH1), output (CH2) and response PSDs

In figure 6.5.2 is presented the comparison between a time domain periodic polar NRZ signal with 100kHz (fig.6.5.2a) and 500kHz (fig.6.5.2b). As expected, the last signal is heavily corrupted due to the signal integration and high jitter.

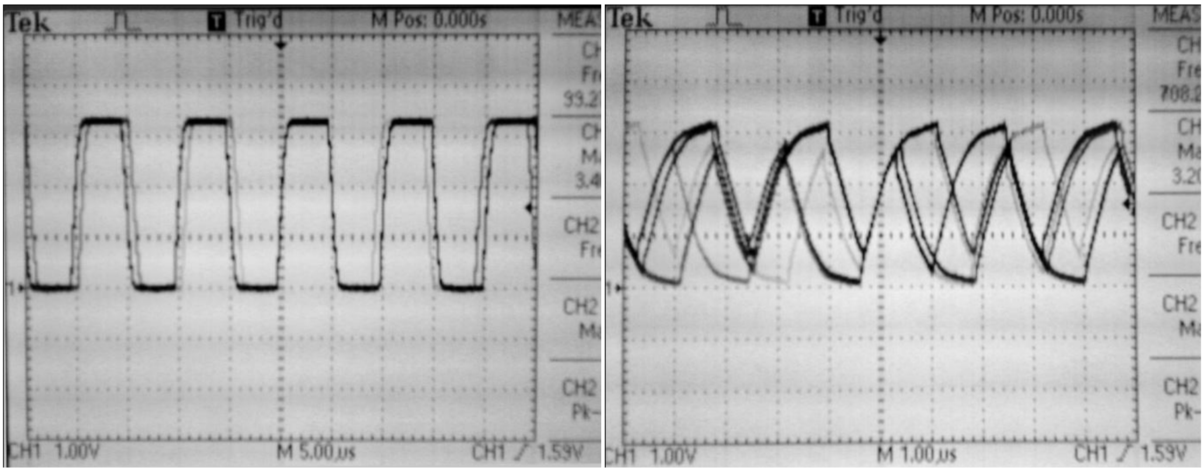
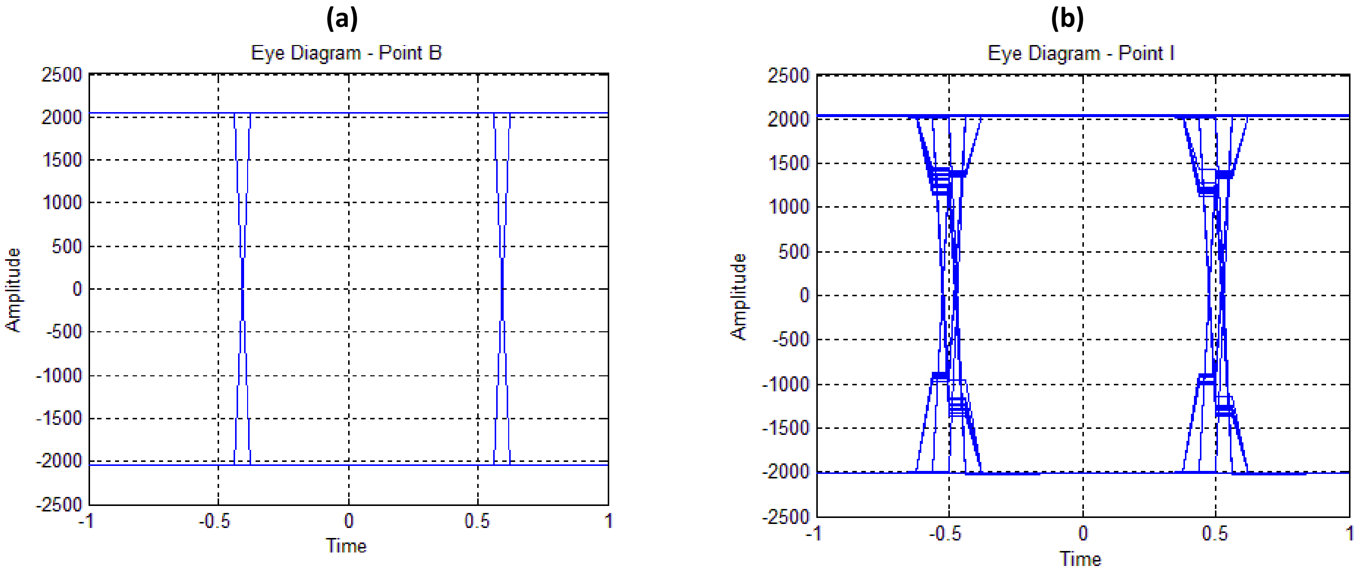


Figure 6.5.2 – Time plots for a periodic signal with a) 100kHz and b) 500 kHz at the DAC output

6.5.2 ADC

In this section only a qualitative ADC tests will be done by the eye diagram analysis. The eye diagrams were obtained with 256 symbols with 16 samples each. The eye diagram depicted in figure 6.5.3a) is the eye diagram at the ADC input. The eye diagrams b), c) and d) were obtained at the ADC output for a random signal generated at 100kHz, 250kHz and 400kHz, respectively. As in the case of the DAC analysis, the signal was formatted with the polar NRZ line coding and sent over a dedicated digital port in the master FPGA before being connected to the ADC input. Despite the signal at 400kHz is below the expected frequency limit for the ADC (735 kHz to Digilent PmodAD1™), it is highly corrupted, due to jitter. Therefore, is advised to use the tested ADC only for signals operating below 300kHz, or replace the ADC by a higher speed one.



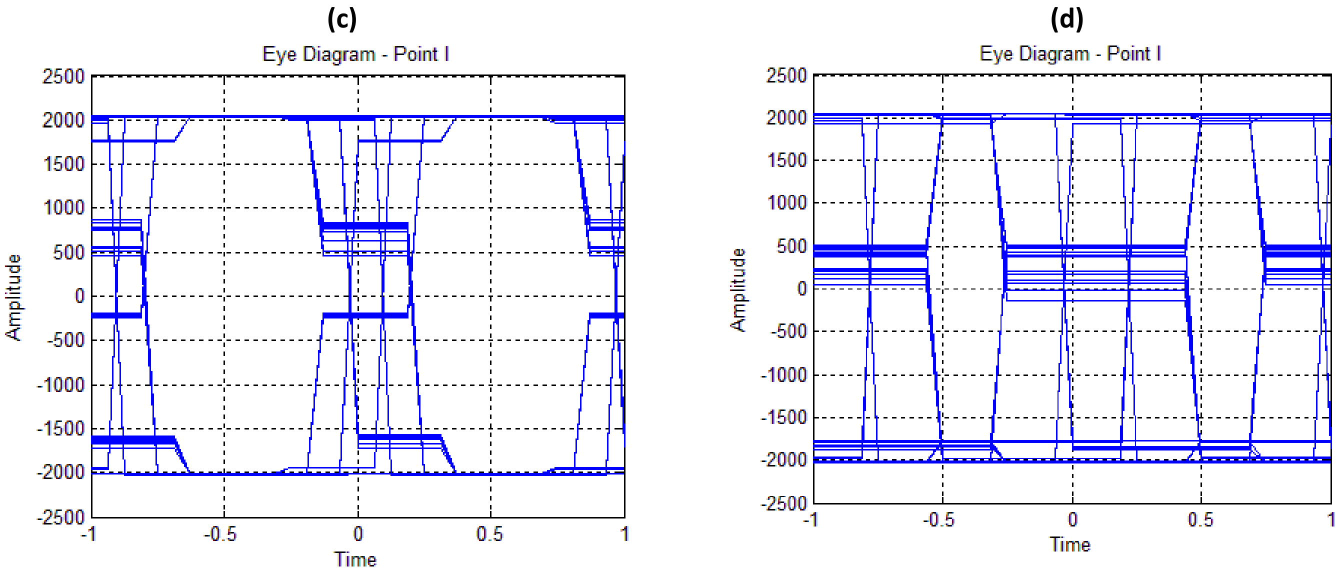


Figure 6.5.3 – Eye diagrams at the a) ADC input and ADC output for signals generated at b) 100kHz, c) 250kHz and d) 400kHz.

6.6 Receiver Filter

For the receiver filter, the user can set the cut-off frequency of a Butterworth low-pass filter with the same proprieties of the one mentioned in section 6.3 of this text. To test the performance of this filter, some other blocks in the system will be used. A random unipolar NRZ signal with $R_s = 10KHz$ was generated and was formatted with a raised cosine filter (with r=0.5) in the transmitter. The resultant signal has a bandwidth equal to $B_T = \frac{(1+r).R_s}{2} = 7{,}5KHz$ and was presented in section 6.2.2. In the channel, will be also used the AWGN generator with a standard deviation of 0.25 (bilateral PSD of $-31{,}07\ dBm/Hz$). The receiver filter was set to a -3dB cut-off frequency of 7,5 KHz. The diagrams in these points are presented in figure 6.6.1. The DAC and ADC used were the ones tested in section 6.5.1 and 6.5.2, respectively.

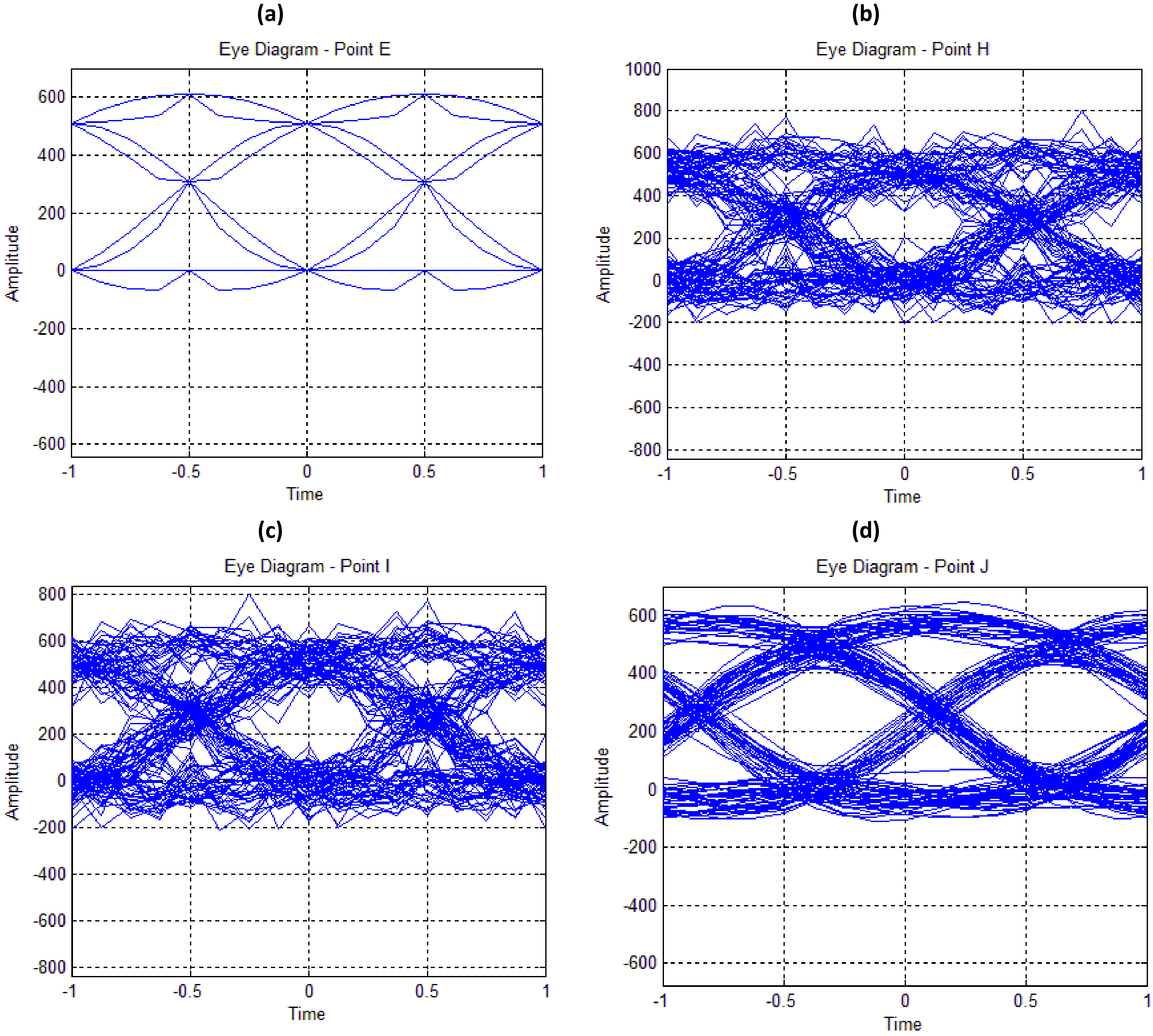


Figure 6.6.1 – Unipolar NRZ eye diagram after a) raised cosine filter b) AWGN c) receiver ADC d) receiver filter.

As expected, the DAC and the ADC influenced the signal. It can be observed the noise reduction at eye diagram due to the receiver filter (fig. 6.6.1d). The filtered signal in the receiver has also a high jitter and a delay in comparison with the transmitted signal (fig. 6.6.1a), which can lead to decision errors. However, it has an acceptable ISI and noise margin.

6.7 Data Recover

In order to avoid signal distortions from the DAC and ADC peripherals, a unipolar signal of $R_b = 10\text{ k bits/s}$ was generated in the transmitter (in master FPGA), formatted by the raised cosine filter and sent to the slave FPGA (receptor block). The maximum level that the signal can take is equal to $V^{+} = 2047$ (signal factor equal to one). Noise was added at the formatted raised cosine signal and filtered by the receptor filter block. Each error probability value is determined through the analysis of 32000 samples, and by their comparison with the original generated data. Therefore, the user must select simultaneously the original data signal (channel 'A') at the master FPGA and the recovered data signal (channel 'O') at the slave FPGA. In the common user manual presented in appendix A is shown a graphic presentation of the user interface, for further guidance on the 'A' and 'O' nomenclature.

6.7.1 Receptor filter cut-off frequency influence

For this first experiment, the raised cosine filter has a roll-off of 50% and the noise has a bilateral P.S.D equal to -31.07 dBm/Hz. The receptor filter cut-off frequency will change between 4,6kHz and 31kHz, as represented in figure 6.7.1. The decision threshold was set to the middle of the signal range (0.5), which for the studied case is the optimum threshold. The sampling phase was set to 2 and the clock recovery block was working with a memory of 8 positions.

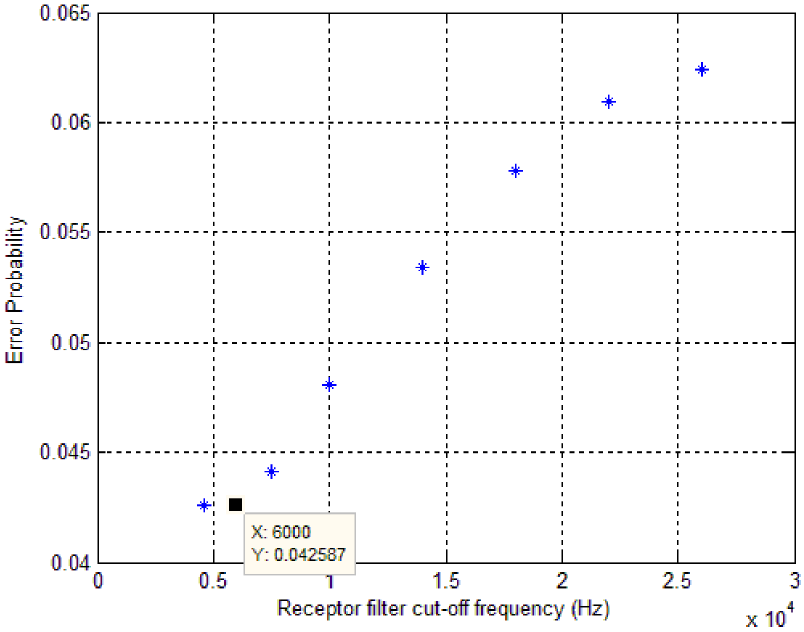


Figure 6.7.1 – Error probability dependency with receptor filter cut-off frequency

As depicted in figure 6.7.1, the minimum error probability occurs for a cut-off frequency equal to 6kHz, which is close to the signal bandwidth (7.5kHz) defined by the raised cosine filter at 50%. The receptor filter does not introduce significant ISI for a cut-off frequency of 6kHz, neither it will allow an excessive amount of noise at the decision circuit input.

6.7.2 *Sampling Phase influence*

The receiver performance with the sampling instant was tested for a unipolar RZ line coding and a post detection filter cut-off frequency of 10kHz. The sampling factor will be changed to 16 and sampling instant will vary from 1 to 7, allowing a best analysis of their effect into the error probability. In figure 6.7.2 is presented the effect of the sampling instant into the error probability, where the sampling instant is normalized at the symbol period. Is possible to observe that the minimum error probability occurs when the phase is equal to 1 (0,071.T_b). It is important to mention that the sampling process introduces significant errors for a phase equal to 0 and it was not considered in plot. The error probability increase verified in this graphic (in comparison with the one presented in figure 6.7.1) was due in part to the sampling frequency increase from 80kHz to 160kHz, which in consequence increases the DAC – ADC attenuation effect into the data signal.

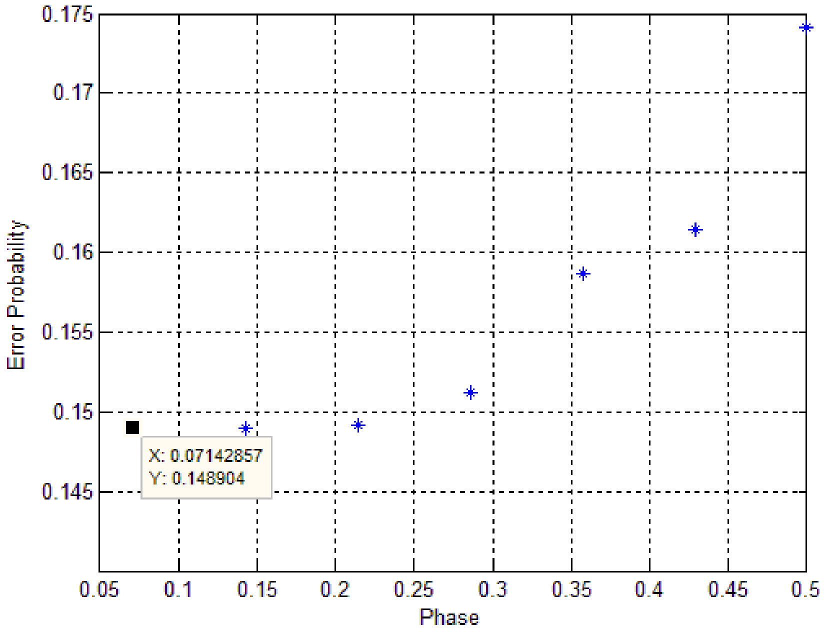


Figure 6.7.2 – Error probability dependency with the sampling instant normalized at the bit period

6.7.3 *CRC Memory size influence*

Through the measurements taken from an unipolar NRZ signal (table 6.7.1) with the system conditions presented in section 6.7.1. is possible to infer that the memory size will not strongly influence the recovered data signal error probability. However, for larger memory sizes, the recovered data signal will be less vulnerable to unexpected width changes (due for example to noise), but it will take more time to adapt to expected signal increase or decrease.

Memory Size	Error Probability
2	0,0426
4	0,0429
8	0,043
16	0,0426
32	0,0428

Table 6.7.1 – Error probability dependency with the memory size

6.7.4 Decision thresholds influence

In equation 3.4.3 was presented the optimum threshold voltage for a Gaussian noise with a mean value, μ, that adds to the symbol voltages V_1 and V_0. In this experiment the signal has a normalized high level $V_1 = 1$ and a lower level $V_0 = 0$. The optimum threshold is determined in equation 6.7.1 for these conditions, which results into 0.5.

$$\mathrm{v_{th}} = \frac{(V_1+\mu)^2 - (V_0+\mu)^2}{2(V_1 - V_0)} = 0{,}5 \tag{6.7.1}$$

The plot presented in figure 6.7.3 corroborates the theoretical value, since it is for this voltage threshold that the error probability is minimum. Since the noise applied to the data signal can be constructive or destructive, for decision thresholds higher or lower than 0.5, will lead to a bigger error probability.

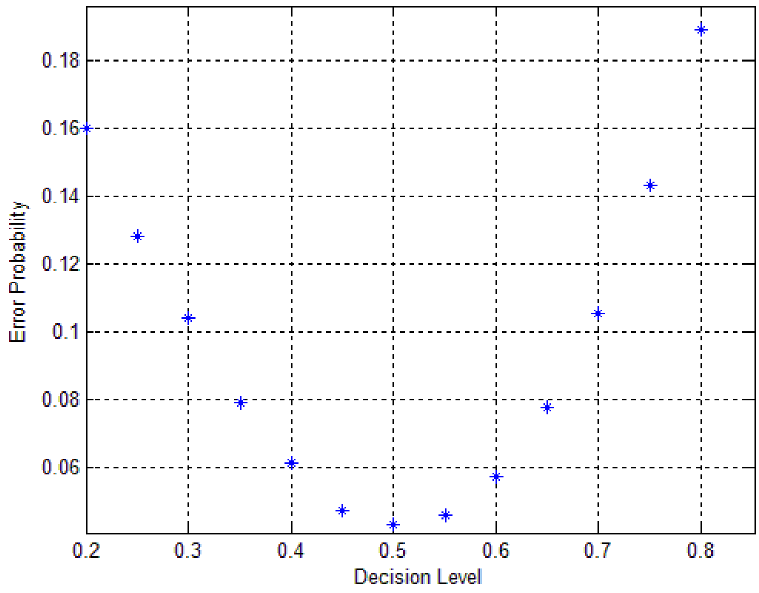


Figure 6.7.3 – Error probability dependency with the decision level threshold

7. Conclusions and Future Work

During this work were analyzed different functional blocks of a baseband digital communications kit for training and research purposes. This kit is based on two FPGAs which communicate via PC for initialization and data analysis proposes. A Matlab interface was developed, which allows the common user to interact with the system in an intuitive way, as presented in appendix A.

This system is made out of separated blocks, which will allow an easy communication between them and the development of new functionalities oriented for new components, simulation methods and control mechanisms. Other advantage of this system is its ability to simulate, within the same physical components, a large variety of configurations. However, as concluded in chapter 6, the communication chain performance is severely attenuated due to the usage of a pair DAC and ADC to make the communication between both FPGAs. Therefore, one of the major modifications that must occur in this system goes through the replacement of the current ADC and DAC for new devices with a higher bandwidth.

In the future, the band pass signal modulation, can be implemented in the current communication chain framework, comprising the basic digital transmission systems based on ASK, FSK and PSK modulation formats to more complex transmission systems based on M-ary Orthogonal Signals and coherent detection. The FIR filters should be redesigned so that they meet the real digital communications filters, as mentioned in the end of chapter 6.2.2. New filters types can also be added to the system, such as the optical cable and free space in the channel filter. This increase in complexity may require the usage of FPGAs with more logic resources, namely more DSP slices, which are used in the filters IPcores.

As seen in appendix C, the kit developed has a fragile layout, which makes it unsuitable for class usage as it is. Therefore, a cover case must be used to protect the system from unwanted usage.

Appendix A - User Manuals

Programmer Manual

Commands List

In order to the user interact with the system in a lower level than the Matlab guide interface, are presented in the table A.1.1 the commands that are needed to initialize it. All commands have the following structure ">ABC.DE.Data\r", where each letter represents a byte. Each set of letters is separated by a dot for reading convenience. The first three letters translate the block in the system (for example the sequence generator, FIFO, DAC, etc) and the second set of two letters represent the specific parameter of that block (for example the number of cells in the sequence generator block). The data field size depends on the type of parameter, and can have decimal digits and/or commas and signal bytes ('-', '+' – represented by the 'S' letter in table A.1.1). The digits are represented by the letters D2 and D1 which can take the sets {0,1} and {0,1,2,3,4,5,6,7,8,9}, respectively. The orange commands are "pseudo-commands" since they can only be found inside the Matlab guide programming being then translated to the black or green commands by Matlab. The green commands, on the other hand can only be found the PC – master FPGA communication. The black commands are present in all system chain, from the Matlab guide environment until the FPGAs.

Chain Position	Block Description	Parameter Description	Digit Format	Initialization character	Module Identifier	Separation character	Parameter Identifier	Separation character	Number of digits	Terminator
(1)Transmitter	Sequence Generator	Number of Cells	D1	>	**TSG**	.	**NC**	.	**2**	**\r**
(2)Transmitter	Sequence Generator	Seed	D1	>	**TSG**	.	**SD**	.	**10**	**\r**
(3)Transmitter	Sequence Generator	Clock	D1	>	**TSG**	.	**CL**	.	**8**	**\r**
(4)Transmitter	Sequence Generator	Select	D1	>	**TSG**	.	**SL**	.	**1**	**\r**
(5)Transmitter	Sequence Generator	Mode	D1	>	**TSG**	.	**MD**	.	**1**	**\r**
(6)Transmitter	Sequence Generator	Number of Words	D1	>	**TSG**	.	**NW**	.	**6**	**\r**
(1)Transmitter	FIFO & DAC Input selection	Channel 1	D1	>	**TDC**	.	**C1**	.	**5**	**\r**
(2)Transmitter	FIFO & DAC Input selection	Channel 2	D1	>	**TDC**	.	**C2**	.	**5**	**\r**
Transmitter	FIFO Input Enable	Channel 1	D1	>	**TFF**	.	**C1**	.	**1**	**\r**
Transmitter	FIFO Input Enable	Channel 2	D1	>	**TFF**	.	**C2**	.	**1**	**\r**
(1)Transmitter	Pulse Shaping	Line coding	D1	>	**TPS**	.	**LC**	.	**2**	**\r**
(2)Transmitter	Pulse Shaping	Signal Factor	S D2 , D1 D1 D1	>	**TPS**	.	**SF**	.	**6**	**\r**
(3)Transmitter	Pulse Shaping	Duty Cycle Enable	D1	>	**TPS**	.	**DE**	.	**1**	**\r**

(4)Transmitter	Pulse Shaping	Duty Cycle	D1	>	TPS	.	DC	.	3	\r
(1)Transmitter	Filter	Sampling Factor	D1	>	TFL	.	SF	.	3	\r
(2)Transmitter	Filter	Type	D1	>	TFL	.	TP	.	1	\r
(3)Transmitter	Filter	Roll-Off (%)	D1	>	TFL	.	RO	.	3	\r
(4)Transmitter	Filter	Cutt-Off Frequency	D1	>	TFL	.	CO	.	8	\r
Transmitter	Filter	Coefficients	S D2 , D1 D1 D1	>	TFL	.	CF	.	6×23=138	\r
(1) Channel	Filter	Type	D1	>	CFL	.	TP	.	1	\r
(2) Channel	Filter	Cutt-Off Frequency	D1	>	CFL	.	CO	.	8	\r
Channel	Filter	Coefficients	S D2 , D1 D1 D1	>	CFL	.	CF	.	6×23=138	\r
(1) Channel	Noise	Standard Deviation	S D2 , D1 D1 D1	>	CNS	.	SD	.	6	\r
(1) Receptor	FIFO & DAC Input selection	Channel 1	D1	>	RDC	.	C1	.	5	\r
(2) Receptor	FIFO & DAC Input selection	Channel 2	D1	>	RDC	.	C2	.	5	\r
Receptor	FIFO Input Enable	Channel 1	D1	>	RFF	.	C1	.	1	\r
Receptor	FIFO Input Enable	Channel 2	D1	>	RFF	.	C2	.	1	\r
Receptor	Filter	Coefficients	S D2 , D1 D1 D1	>	RFL	.	CF	.	6×23=138	\r
(1) Receptor	Filter	Type	D1	>	RFL	.	TP	.	1	\r
(2) Receptor	Filter	Cutt-Off Frequency	D1	>	RFL	.	CO	.	8	\r
(1) Receptor	Decision	Upper threshold	S D2 , D1 D1 D1	>	RDS	.	UP	.	6	\r
(2) Receptor	Decision	Lower threshold	S D2 , D1 D1 D1	>	RDS	.	LO	.	6	\r
(1) Receptor	CRC	Memory Size	D1	>	RCR	.	MS	.	3	\r
(1) Receptor	Sampling	Phase	D1	>	RSP	.	PH	.	3	\r
(1) System	Plots	Time	D2	>	SPS	.	TM	.	4	\r
(2) System	Plots	Frequency	D2	>	SPS	.	FR	.	4	\r
(3) System	Plots	Eye Diagram	D2	>	SPS	.	ED	.	4	\r

Table A.1.1 – System Commands

Add new command

This command based system allows the user to add blocks more easily in the communication chain. After name the new command based on the type of block and parameters, it must be sent first for the microprocessor in master FPGA. In order to do it, is possible to use the Matlab in the PC. However, to the processor accept the command as a valid one and their data value be witted in the block data word, the following actions must be done:

1. Access the SDK workspace of MFPGA, more precisely the file “MFPGA.c”;
2. Increase the variable *NMaxCommands* to accommodate the number of the newly added commands;
3. Still in the .c file, add to the *LOAD COMMANDS* section the “if” conditions required to the processor accept that command;
4. In the newly “if” condition created, store the command data in a new array variable. The size of this variable will depend on the number of digits/signal characters in this field. For example the command “TSG.NC” will only carry 2 digits of data, since the maximum number of cells in the sequence generator is 32. Therefore, the array variable “TSG” only need to have 2 positions;
5. Still in the “if” condition, the function *COMMANDLOOP* or *COMMANDSENT* needs to be added if the given command is only for initialize the block data word in master FPGA or it will be sent also to slave FPGA, respectively;
6. In the *BLOCKS INITIALIZATION* section transform the data that was stored in the array variable so that it can be used to initialize the respective block in the FPGA. For example, if the command “TSG.NC” brings the data “12”, the first digit needs to be multiplied by 10 and added with the second digit, before writing the value in the data word 0.
7. The writing process is finalized through the use of the function *REGISTER_WRITE*. Place the data word writing before the resetting clock process;
8. If the commands were sent to the slave FPGA, the procedure above, except the step 5, must be repeated.

After the block data word has been written, the information in it must be read in VHDL. The only standard procedure that must take place in VHDL code is to create in the “user_logic.vhd” file (“chain.xise” project) the new signal in the form “s_auxMem_x” where “x” represent the number of the block data word. For example, to the “TSG.NC” command, the “x” equals 0.

After the procedure mentioned above, the programmer needs to adapt the data to the given block specifications.

Common User Manual

In figure A.2.1 is presented the Matlab user interface layout. This menu pop’s up after running the *menu.m* file. Each one of the blocks mentioned in this document is depicted in this interface with a set of boxes, which represent their input parameters. Each numeric parameter range is presented in table A.2.1

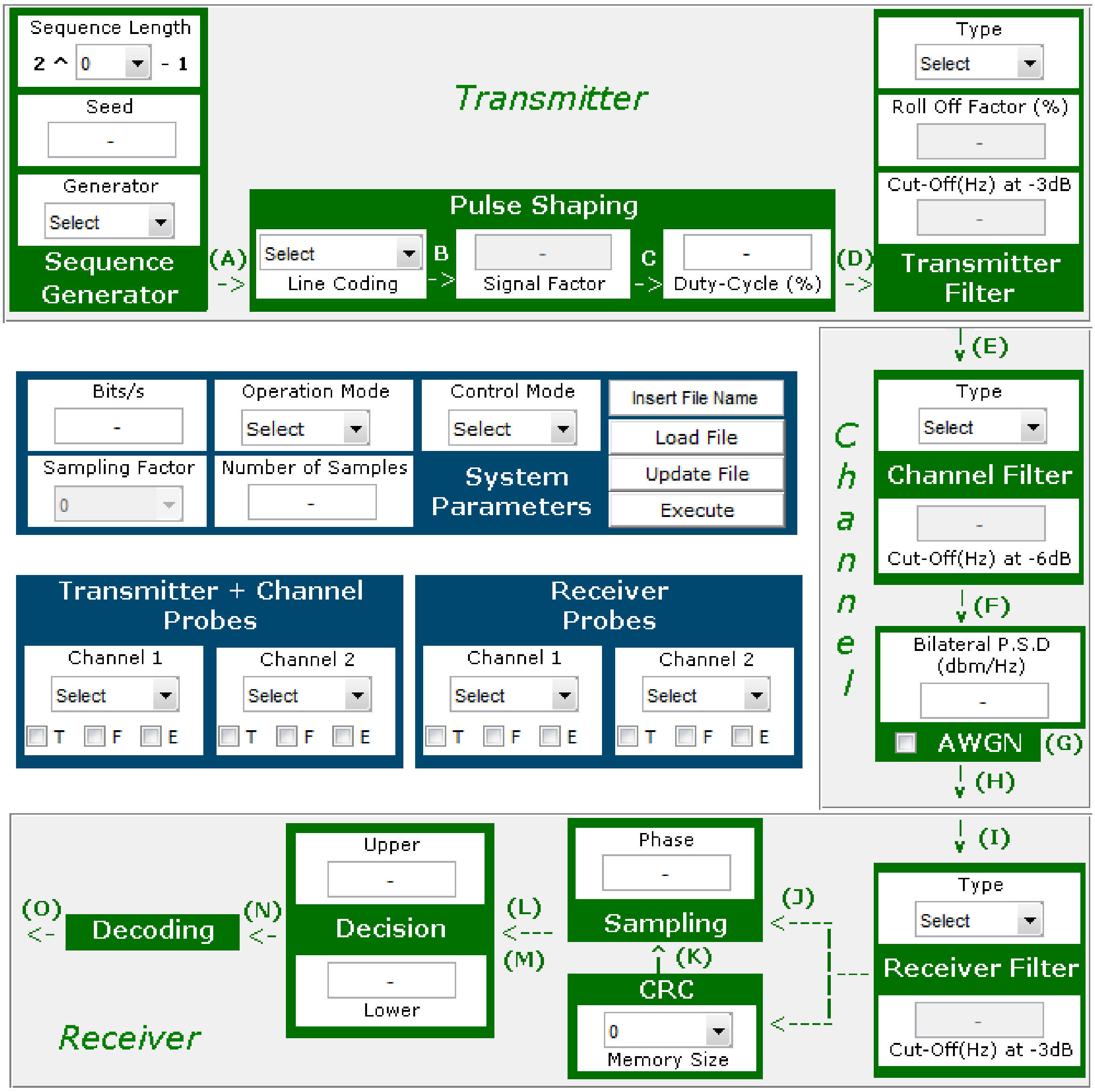


Figure A.2.1 – Matlab user interface layout

Parameter	Range
Sequence Length	[1,32]
Seed	[1, 2147483647]
Signal Factor	[0,1]
Duty-Cycle	[0, 100]
Roll-off Factor	[0, 100]
Cut-off(Hz) at -3dB	$\left[\frac{0.112704 \times Sampling\ Factor \times Bit\ Rate}{2}, \frac{0.78927 \times Sampling\ Factor \times Bit\ Rate}{2}\right]$

Cut-off(Hz) at -6dB	$]0,\frac{Sampling\ Factor \times Bit\ Rate}{2}[$
Cable Length (m)	]0,500000]
Bilateral P.S.D (dbm/Hz)	$10log\left(\frac{stdDev \times 1000}{Fs}\right), stdDev \in]0,1]$
Phase	$\left[0,\frac{Sampling\ Factor}{2}\right]$
Upper Decision	[0,1[
Lower Decision	]-1,0]
Bits/s	[0,3000000]
Number of Samples	[1,65536]

Table A.2.1 – Parameters range of Matlab user interface

Initially all the boxes are empty/unset, as shown in figure A.2.1. The chain is represented in the clockwise direction, in which the transmitter is on the top of the figure, the channel in the right side and the receptor in the bottom. In the menu center are presented the system controls and the probes selection, which are transversal to both FPGAs.

System Parameters

The control mode will control two input boxes: the signal factor and the sampling factor. They will only be writable if the control mode is set to *Advanced*. On the other hand the *Step-by-Step* operation mode will set to Bits/s box at the recommended value of 10000 bits/s.

This Matlab interface also has the ability to use a file (*.txt* or *.dat*) to load and update the parameters set by the user. If the user presses the *Update File* button and the specified file does not exist, it will be created.

After all the parameters in the menu are set, is possible to execute the system, pressing the *Execute* button. However, is important to mention that the system will only run if both FPGAs are already configured and the master FPGA UART cable is connected to the PC.

Probes

Since the transmitter and the channel are in the same FPGA, the first two probes presented at the left side, correspond only to these two blocks. For example, for channel 1 or 2 at the Transmitter + Channel probes, one can choose to analyze the signal at one of the points from (A) to (H).

Each probe (also named channel) allows the user to analyze the system through a time, frequency or eye diagram plots, which are selected through the set of the *T*, *F* and *E* boxes, below each channel, as depicted in figure A.2.2. If the user selects simultaneously the time or the frequency plots in both channels, they will appear in the same MATLAB figure. On the other hand, the eye diagram can only be drawn if the data signal has more than 256 symbols.

If the user selects simultaneously the (A) probe at the master FPGA (Transmitter + Channel) and the (O) channel at the slave FPGA (Receiver), a plot will appear, in which the error probability is calculated, through the comparison of the data of both channels. Is important to mention that the signal that is generated at the (A) point suffers a delay throughout the chain which will lead to some samples be discarded to synchronize both signals and determine the error probability. The discarded samples are presented in this plot in the green color.

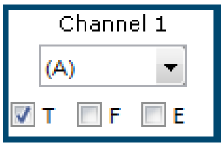
When the user chooses a specific channel, it will also be selected into the probe DACs. However, it can only be used if the operation mode is set to *Continuous*.

Chain

The *Duty-Cycle* input parameter should only be writable if the transmitter filter is not the raised-cosine filter and the selected RZ line coding allows a variable duty cycle. If the raised cosine filter is selected, the *Roll-Off Factor* input box is enabled and the *Cut-Off(Hz) at -3dB* input box is disabled.

As it happens with the transmitter and the receiver filters, when the user sets the *Select* option in the Type box, is not possible to write into the *cut-off* box. On the other hand, to save some space in the menu layout, the *Cut-Off(Hz) at -6dB* box at the channel filter, is also used as the input for the coaxial cable length. The nomenclature of this box will change when the user changes the filter type between the coaxial and the high-pass filter. In all filters, the *None* option will enable to simulate as if the filter was not there (a "short circuit" between its input and output).

The AWGN can be deactivated or activated by unsetting/setting the bilateral P.S.D check box, as depicted in figure A.2.3.

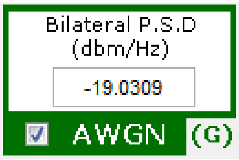


Channel 1

(A)

☑ T □ F □ E

Figure A.2.2 - Master FPGA channel 1 selection



Bilateral P.S.D (dbm/Hz)

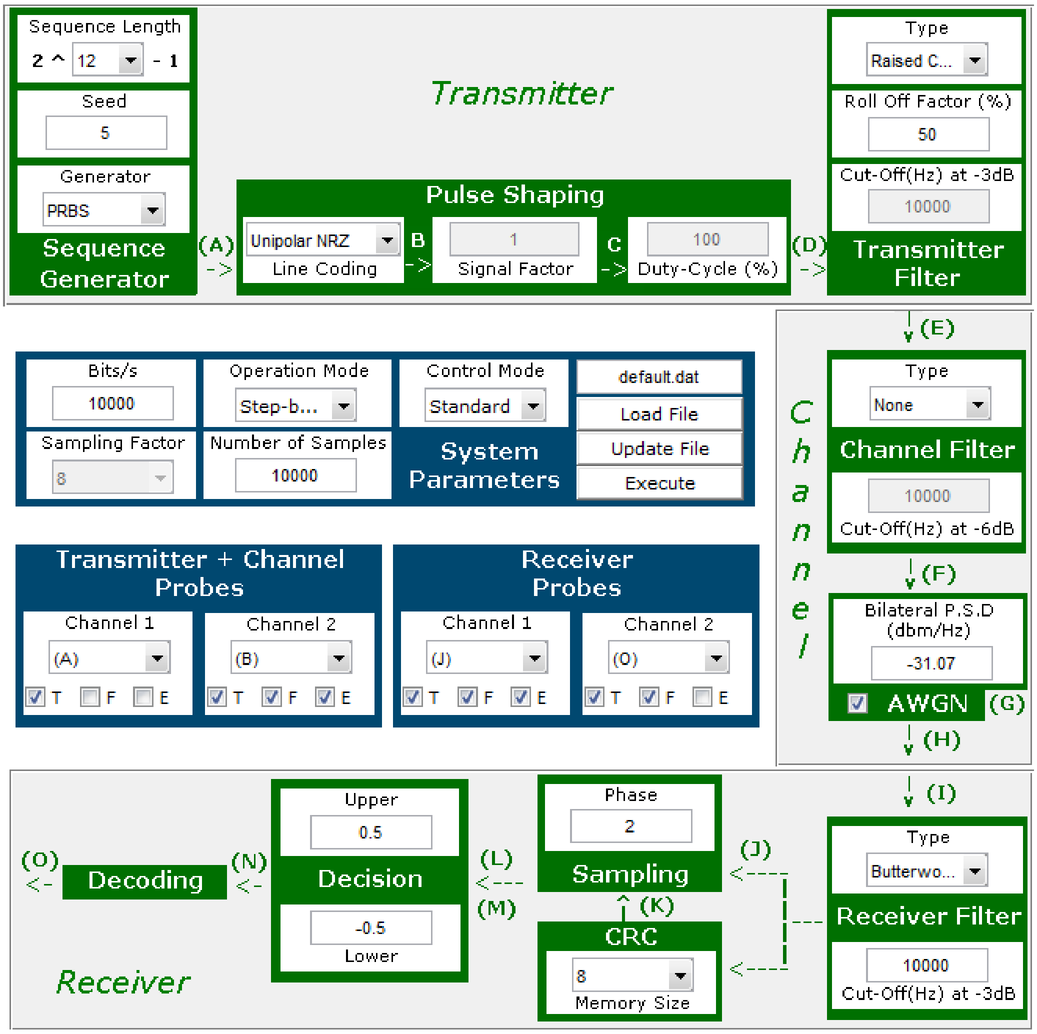
-19.0309

☑ AWGN (G)

Figure A.2.3 – AWGN generation

Getting Started

In this getting started, the parameters contained in the *default.dat* file will be used to show some of the Matlab user interface basic functionalities. Therefore, first off all, they must be loaded into the system through the *Insert File Name* box and by pressing the *Load File* button. The file must be in the same folder as the *menu.m* file. In figure A.3.1, is presented the Matlab user interface layout after the file as been loaded. These parameters were already used in section 6.7 of this document. After the file as been loaded, the master FPGA UART cable must be connected into the PC and the FPGAs must be already running the system files. If the FPGAs are correctly running, a *Power On* message will be presented in the OLED display. To run the system, the *Execute* button must be pressed. After that button is pressed, one must enter in the Matlab command window the com port number of the UART cable. The resulting eye diagrams (point B and J) and error probability plots are presented in figure A.3.2. From them is possible to confirm what has been mentioned in section 6.7: that throughout the communication chain some noise is introduced, leading to the error probability 0,049. The green samples represent the delay that occurs between the original and the recovered signal. From this starting point is now possible to change any parameter or probe in the system to meet the user specifications. To see the selected signals with the probes in the oscilloscope, the probe DACs must be placed into the FPGAs as depicted in the complete system layout presented in appendix C.



Sequence Length 2 ^ 12 - 1
Seed 5
Generator PRBS

Line Coding Unipolar NRZ
Signal Factor 1
Duty-Cycle (%) 100

Type Raised C...
Roll Off Factor (%) 50
Cut-Off(Hz) at -3dB 10000

Bits/s 10000
Operation Mode Step-b...
Control Mode Standard
default.dat
Sampling Factor 8
Number of Samples 10000

Channel 1 (A) ☑ T ☐ F ☐ E
Channel 2 (B) ☑ T ☑ F ☑ E

Channel 1 (J) ☑ T ☑ F ☑ E
Channel 2 (O) ☑ T ☑ F ☐ E

Type None
Cut-Off(Hz) at -6dB 10000

Bilateral P.S.D (dbm/Hz) -31.07
☑ AWGN

Type Butterwo...
Cut-Off(Hz) at -3dB 10000

Phase 2
Memory Size 8

Upper 0.5
Lower -0.5

Figure A.3.1 – Matlab user interface after loading default.dat parameters

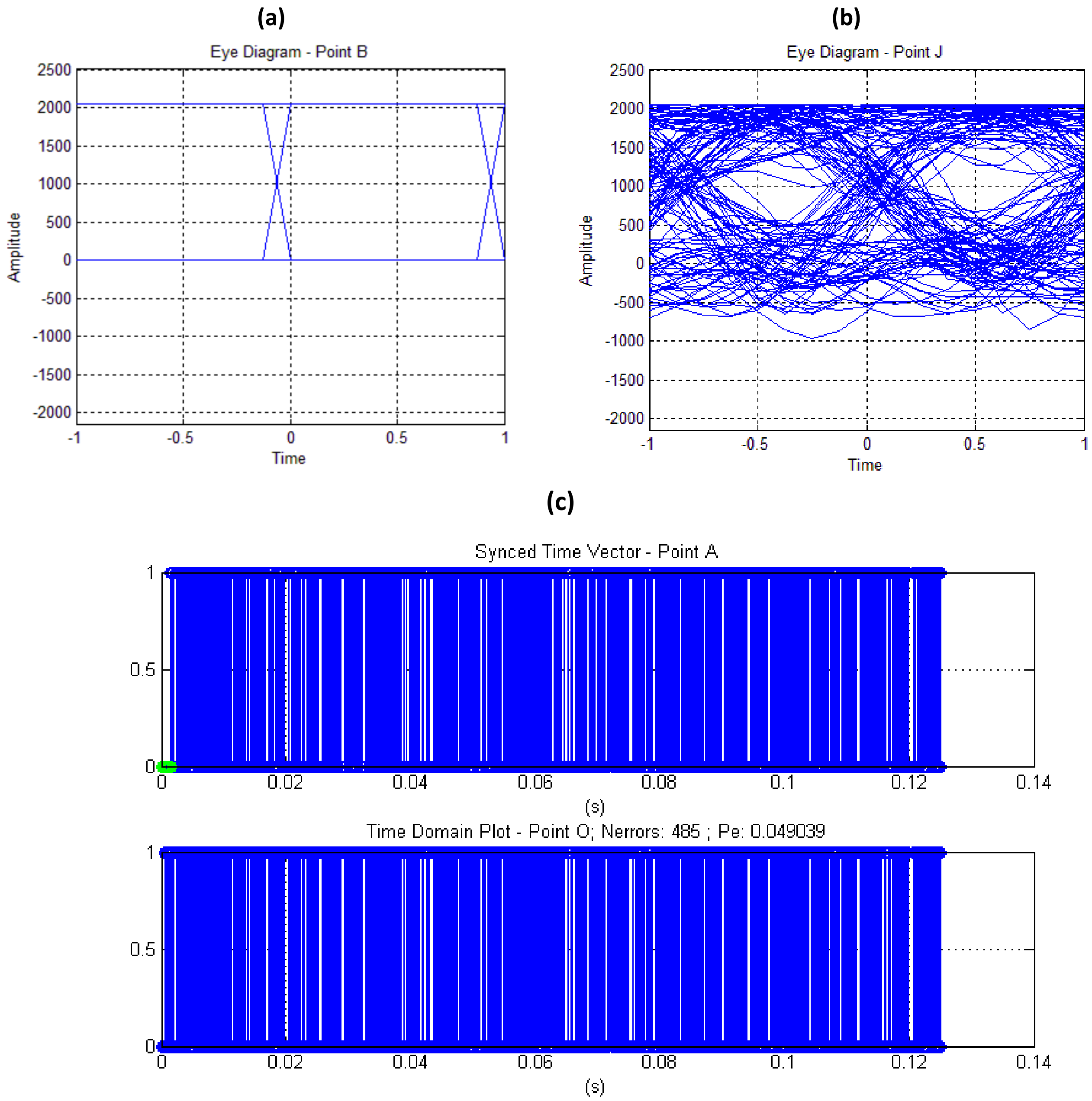


Figure A.3.2 – a) Eye Diagrams and b) Error Probability plots obtained from the *default.dat* file

Appendix B – EDK and IP cores Configuration

EDK

In figure B.1 is presented the block diagram of the embedded development kit, which contains the blocks described in the previous chapters: microblaze, communication chain, control logic and UARTs, among others.

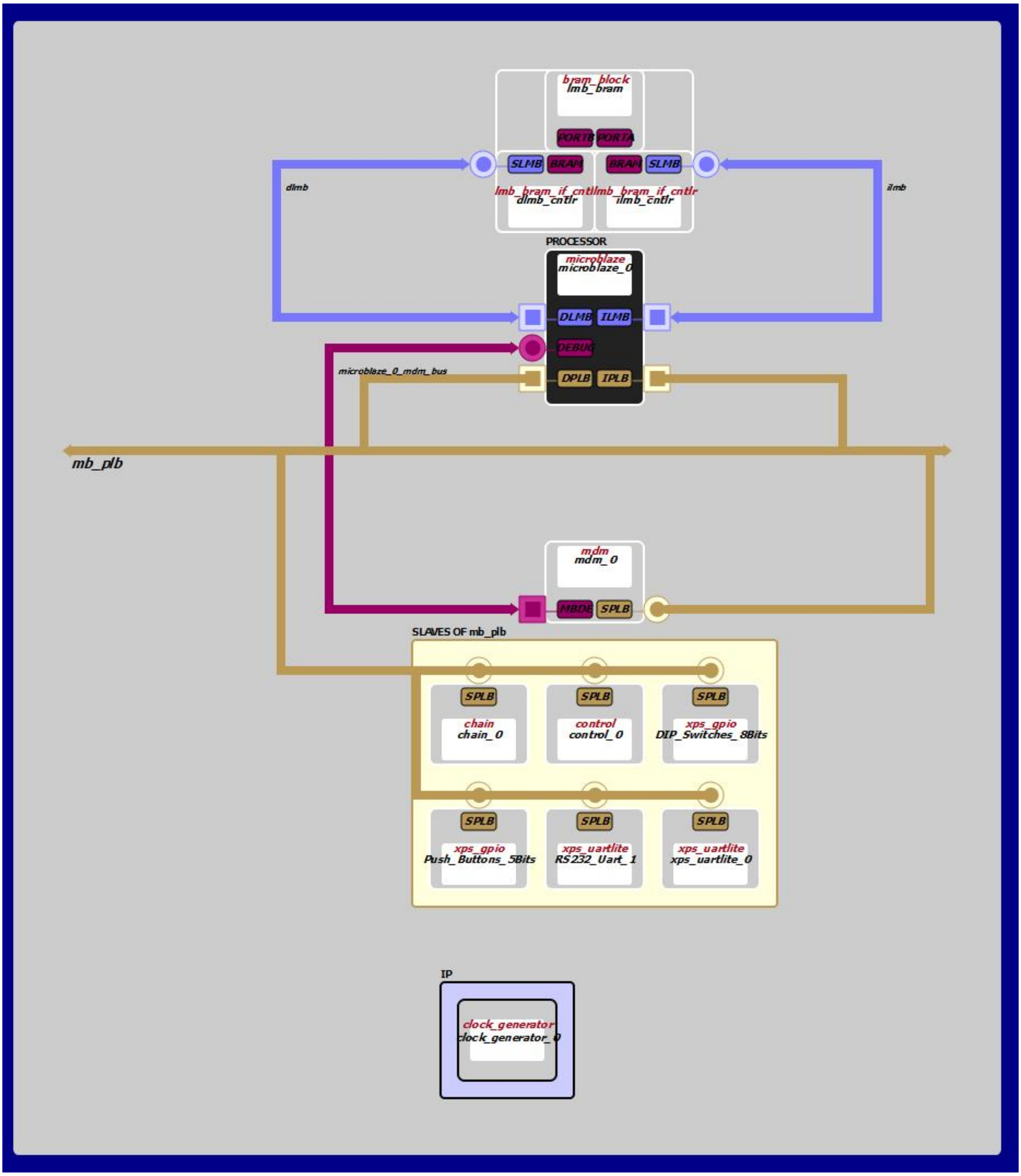
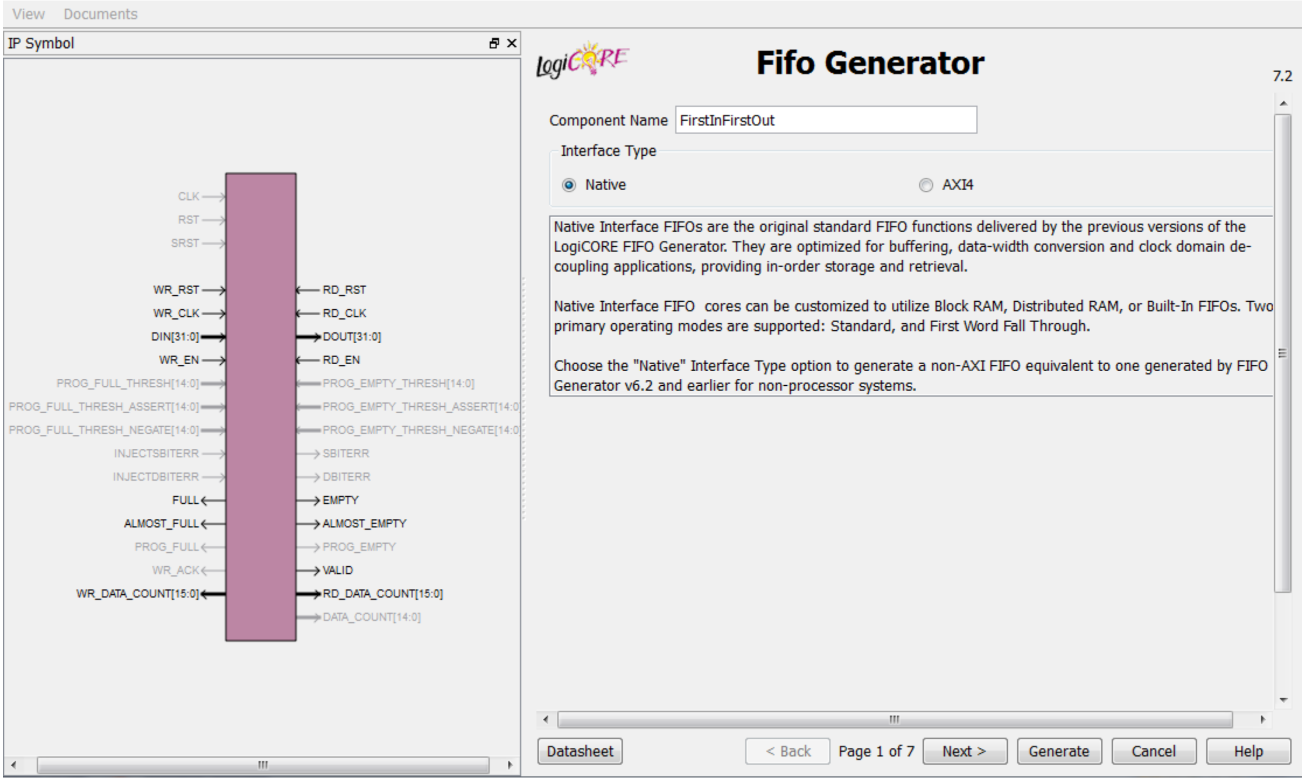


Figure B.1 – EDK Block Diagram

FIFO IP Core

In the process of creating the IP Cores FIFO and Filter, some parameters must be chosen according to the project design. In figures B.2 and B.3 are presented the configuration pages of those IP cores correctly filled.

a)



Component Name FirstInFirstOut

Interface Type

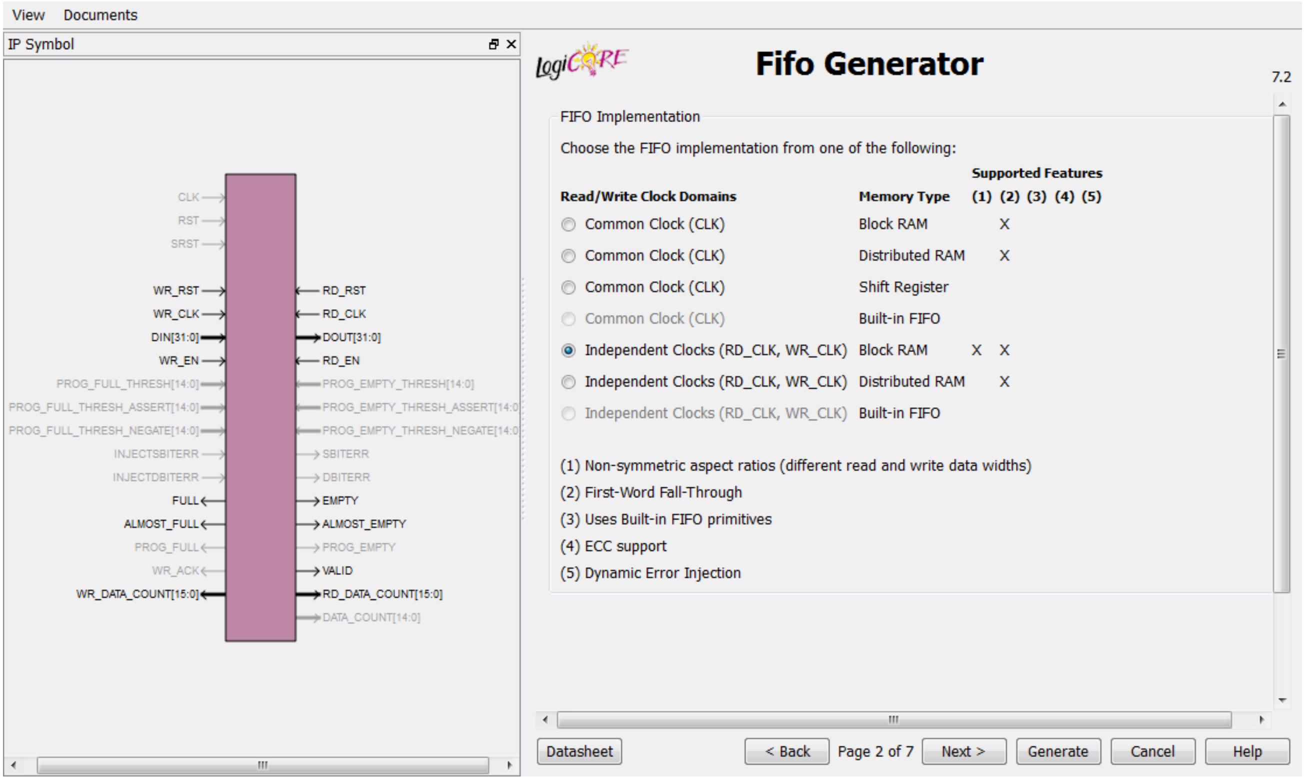
● Native ○ AXI4

Native Interface FIFOs are the original standard FIFO functions delivered by the previous versions of the LogiCORE FIFO Generator. They are optimized for buffering, data-width conversion and clock domain de-coupling applications, providing in-order storage and retrieval.

Native Interface FIFO cores can be customized to utilize Block RAM, Distributed RAM, or Built-In FIFOs. Two primary operating modes are supported: Standard, and First Word Fall Through.

Choose the "Native" Interface Type option to generate a non-AXI FIFO equivalent to one generated by FIFO Generator v6.2 and earlier for non-processor systems.

b)



Fifo Generator 7.2

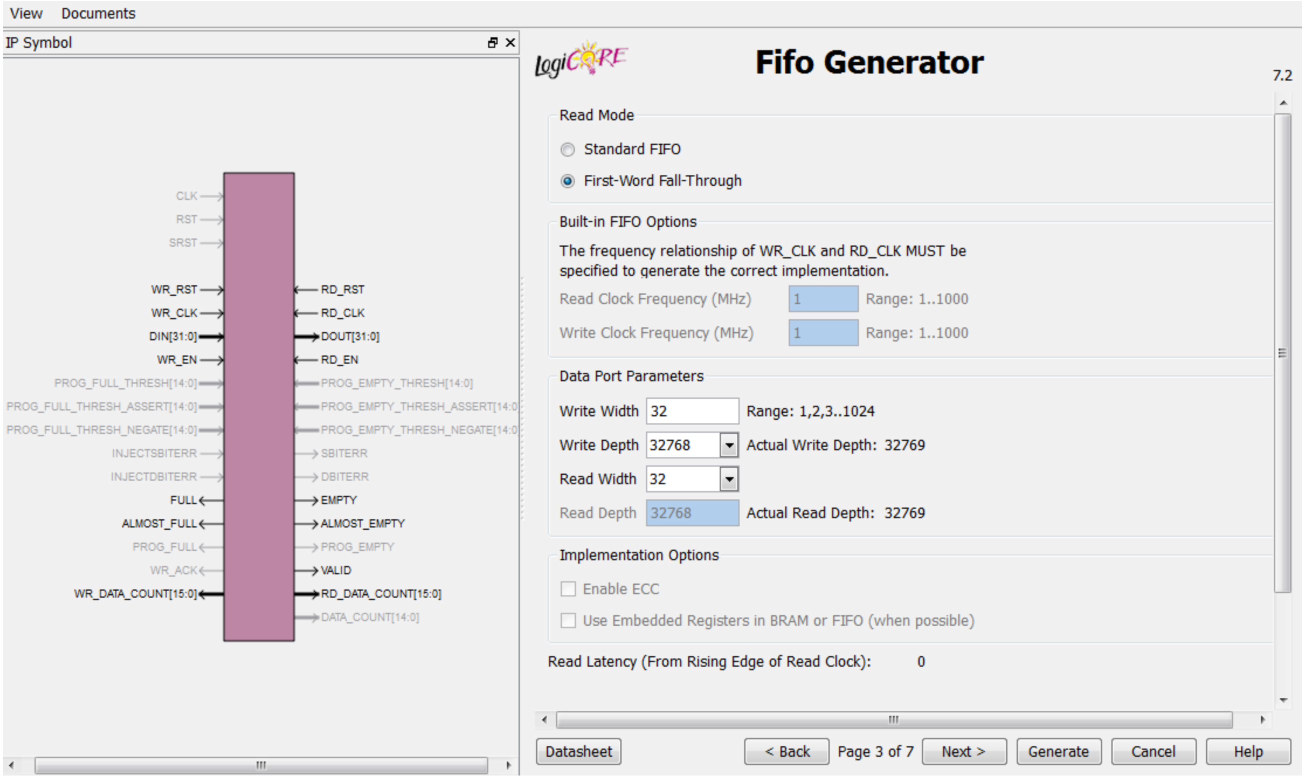
FIFO Implementation

Choose the FIFO implementation from one of the following:

Read/Write Clock Domains	Memory Type	Supported Features (1)	(2)	(3)	(4)	(5)
○ Common Clock (CLK)	Block RAM		X			
○ Common Clock (CLK)	Distributed RAM		X			
○ Common Clock (CLK)	Shift Register					
○ Common Clock (CLK)	Built-in FIFO					
● Independent Clocks (RD_CLK, WR_CLK)	Block RAM	X	X			
○ Independent Clocks (RD_CLK, WR_CLK)	Distributed RAM		X			
○ Independent Clocks (RD_CLK, WR_CLK)	Built-in FIFO					

(1) Non-symmetric aspect ratios (different read and write data widths)
(2) First-Word Fall-Through
(3) Uses Built-in FIFO primitives
(4) ECC support
(5) Dynamic Error Injection

c)



Fifo Generator 7.2

Read Mode

○ Standard FIFO

● First-Word Fall-Through

Built-in FIFO Options

The frequency relationship of WR_CLK and RD_CLK MUST be specified to generate the correct implementation.

Read Clock Frequency (MHz) 1 Range: 1..1000

Write Clock Frequency (MHz) 1 Range: 1..1000

Data Port Parameters

Write Width 32 Range: 1,2,3..1024

Write Depth 32768 Actual Write Depth: 32769

Read Width 32

Read Depth 32768 Actual Read Depth: 32769

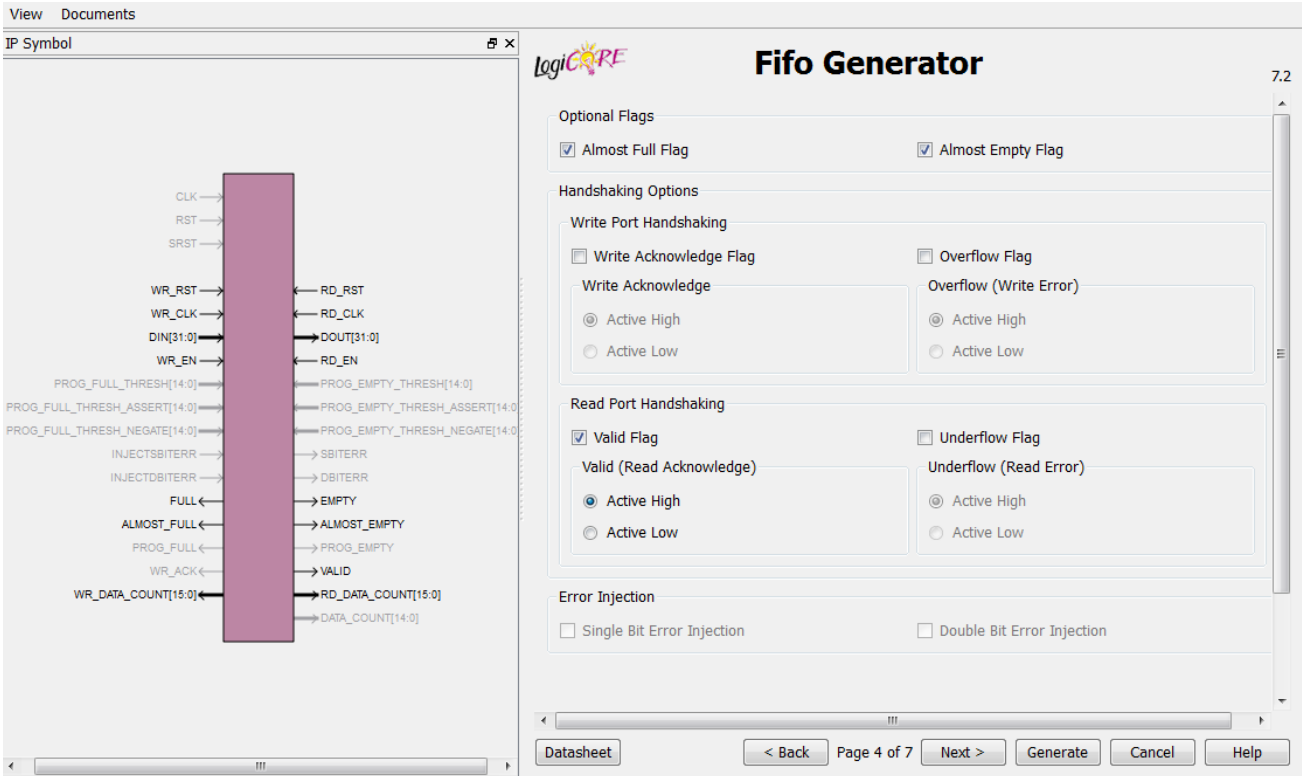
Implementation Options

□ Enable ECC

□ Use Embedded Registers in BRAM or FIFO (when possible)

Read Latency (From Rising Edge of Read Clock): 0

d)



Fifo Generator 7.2

Optional Flags

☑ Almost Full Flag ☑ Almost Empty Flag

Handshaking Options

Write Port Handshaking

☐ Write Acknowledge Flag ☐ Overflow Flag

Write Acknowledge: ● Active High ○ Active Low

Overflow (Write Error): ● Active High ○ Active Low

Read Port Handshaking

☑ Valid Flag ☐ Underflow Flag

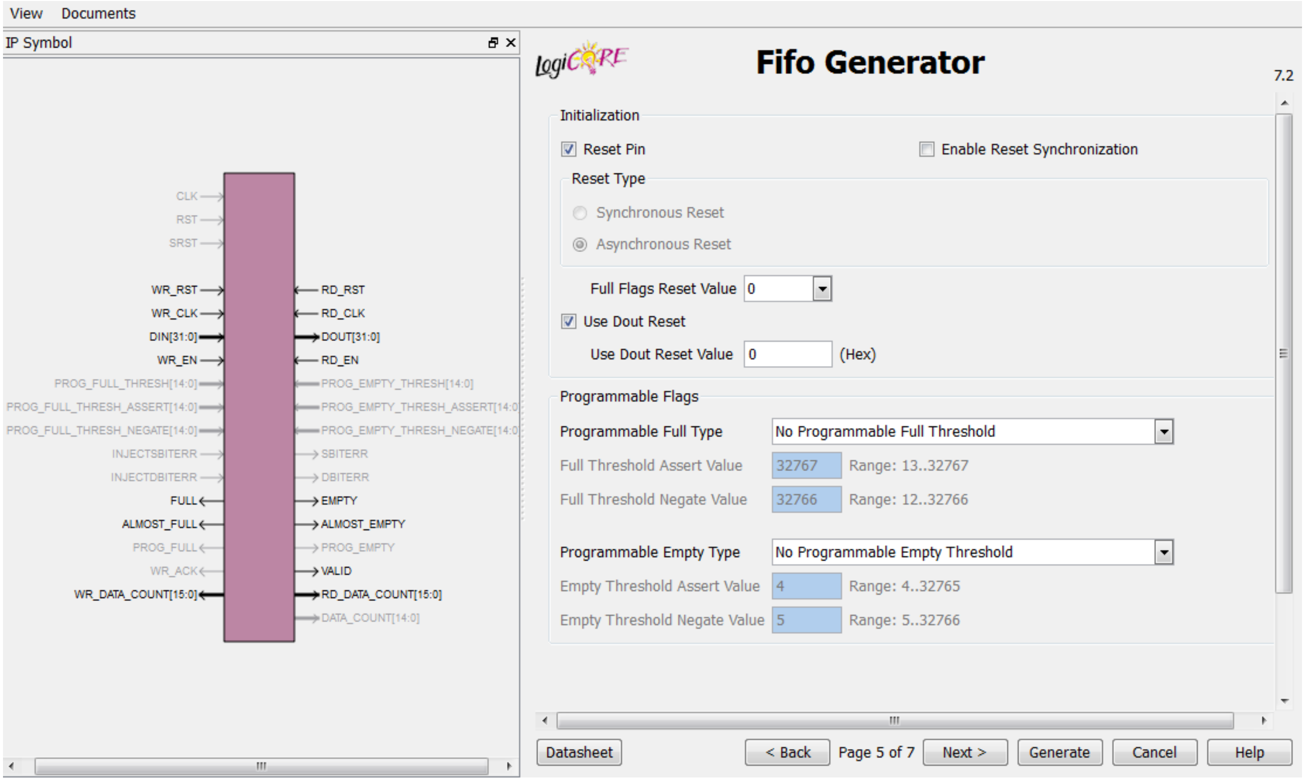
Valid (Read Acknowledge): ● Active High ○ Active Low

Underflow (Read Error): ● Active High ○ Active Low

Error Injection

☐ Single Bit Error Injection ☐ Double Bit Error Injection

e)



Fifo Generator 7.2

Initialization

☑ Reset Pin ☐ Enable Reset Synchronization

Reset Type

○ Synchronous Reset

● Asynchronous Reset

Full Flags Reset Value 0

☑ Use Dout Reset

Use Dout Reset Value 0 (Hex)

Programmable Flags

Programmable Full Type: No Programmable Full Threshold

Full Threshold Assert Value: 32767 Range: 13..32767

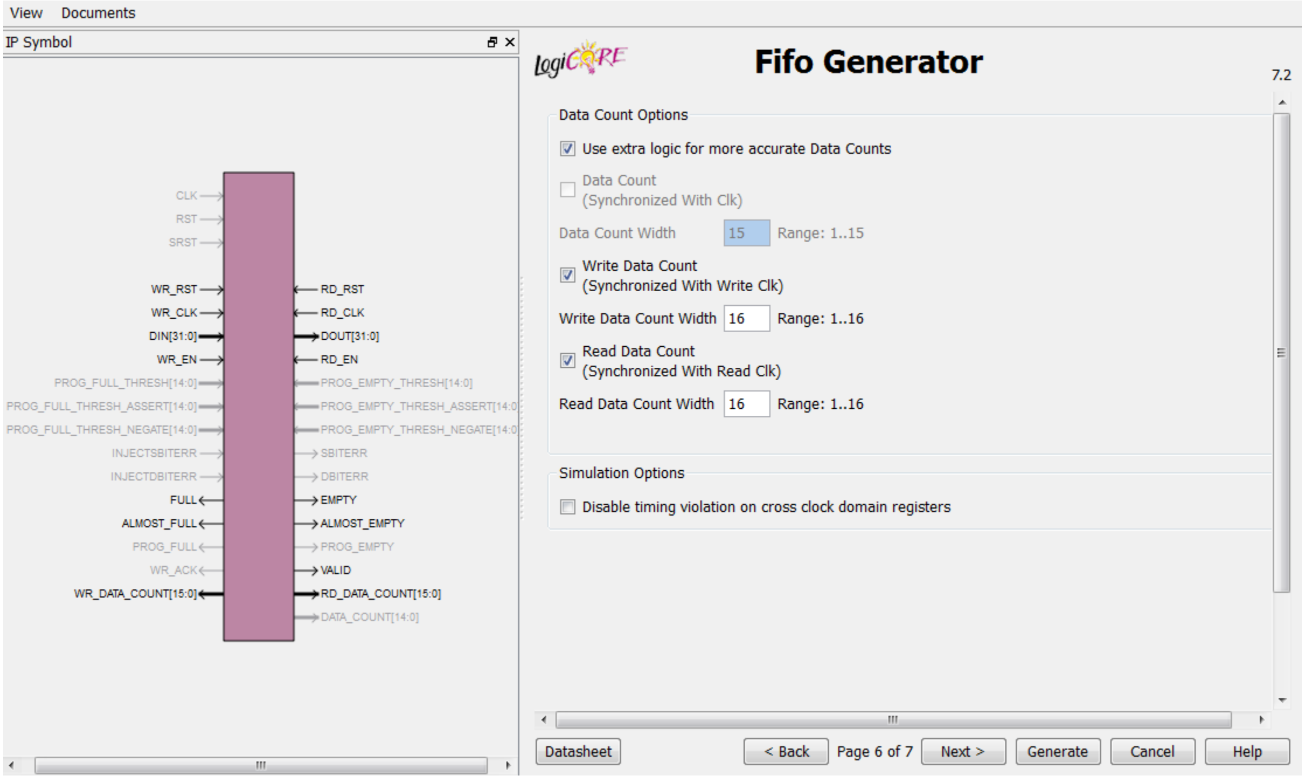
Full Threshold Negate Value: 32766 Range: 12..32766

Programmable Empty Type: No Programmable Empty Threshold

Empty Threshold Assert Value: 4 Range: 4..32765

Empty Threshold Negate Value: 5 Range: 5..32766

f)



Fifo Generator

Data Count Options

☑ Use extra logic for more accurate Data Counts

□ Data Count
(Synchronized With Clk)

Data Count Width 15 Range: 1..15

☑ Write Data Count
(Synchronized With Write Clk)

Write Data Count Width 16 Range: 1..16

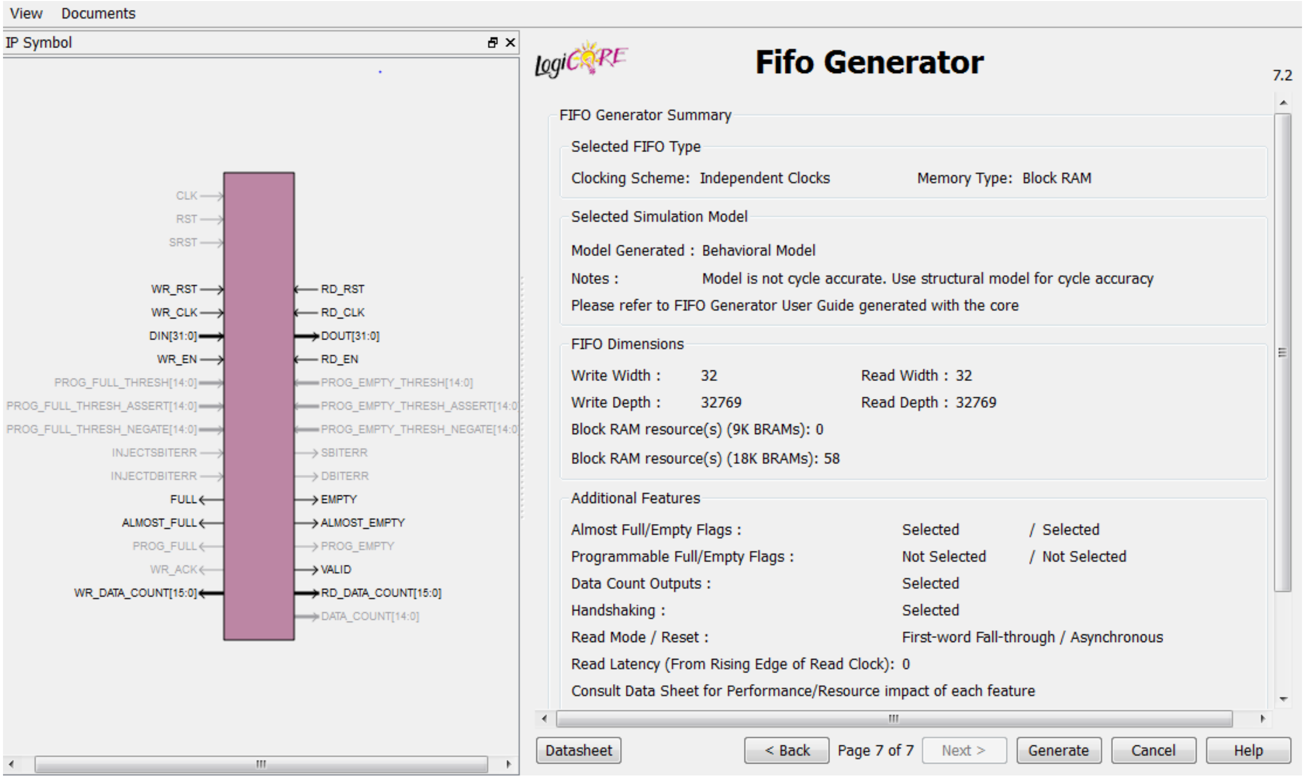
☑ Read Data Count
(Synchronized With Read Clk)

Read Data Count Width 16 Range: 1..16

Simulation Options

□ Disable timing violation on cross clock domain registers

g)



Fifo Generator

FIFO Generator Summary

Selected FIFO Type

Clocking Scheme: Independent Clocks Memory Type: Block RAM

Selected Simulation Model

Model Generated : Behavioral Model

Notes : Model is not cycle accurate. Use structural model for cycle accuracy

Please refer to FIFO Generator User Guide generated with the core

FIFO Dimensions

Write Width : 32 Read Width : 32

Write Depth : 32769 Read Depth : 32769

Block RAM resource(s) (9K BRAMs): 0

Block RAM resource(s) (18K BRAMs): 58

Additional Features

Almost Full/Empty Flags :	Selected	/ Selected
Programmable Full/Empty Flags :	Not Selected	/ Not Selected
Data Count Outputs :	Selected	
Handshaking :	Selected	
Read Mode / Reset :	First-word Fall-through / Asynchronous	

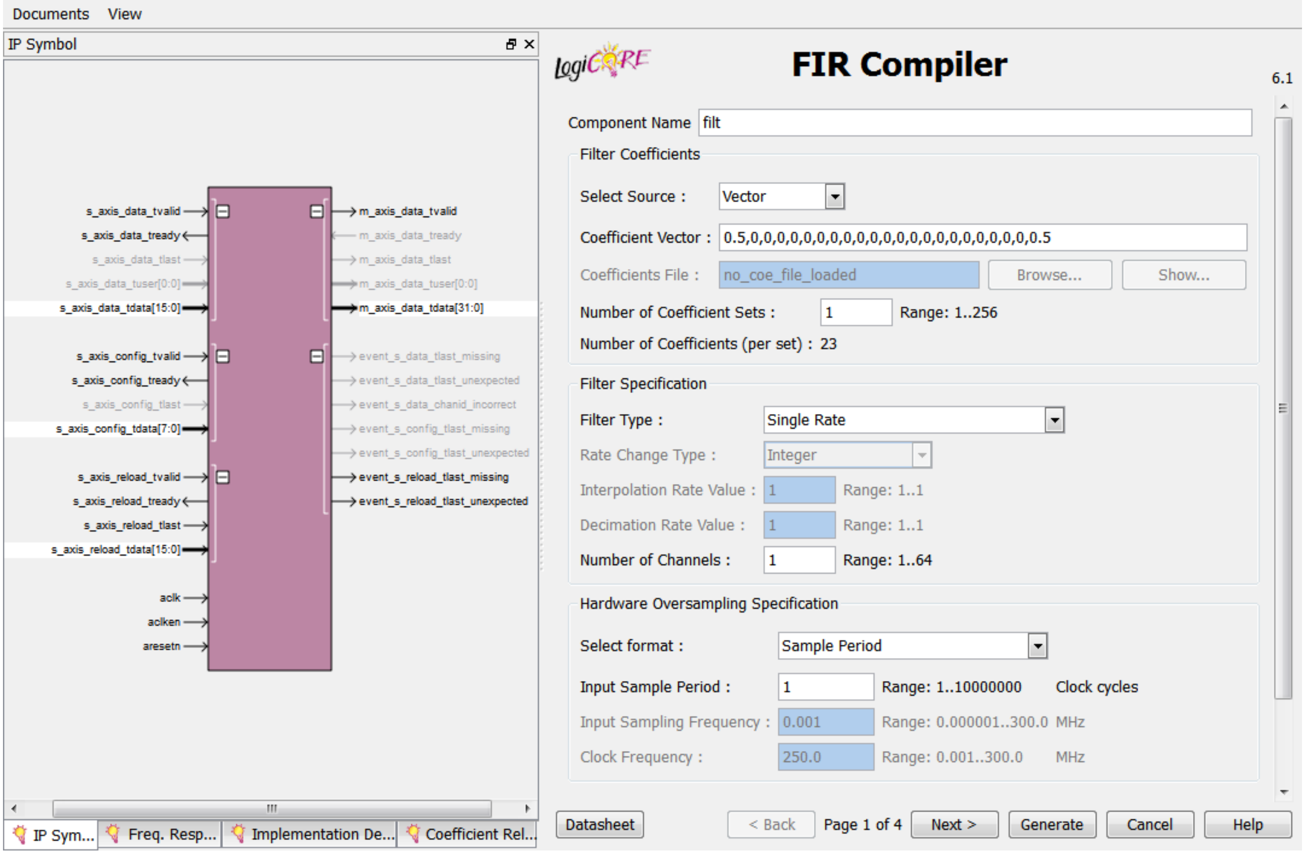
Read Latency (From Rising Edge of Read Clock): 0

Consult Data Sheet for Performance/Resource impact of each feature

Figure B.2 – FIFO IP core parameters configuration. Pages a) 1, b) 2, c) 3, d) 4, e) 5, f) 6 and g) 7

Fir Filter IP Core

a)



FIR Compiler

Component Name filt

Filter Coefficients

Select Source : Vector

Coefficient Vector : 0.5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.5

Coefficients File : no_coe_file_loaded

Number of Coefficient Sets : 1 Range: 1..256

Number of Coefficients (per set) : 23

Filter Specification

Filter Type : Single Rate

Rate Change Type : Integer

Interpolation Rate Value : 1 Range: 1..1

Decimation Rate Value : 1 Range: 1..1

Number of Channels : 1 Range: 1..64

Hardware Oversampling Specification

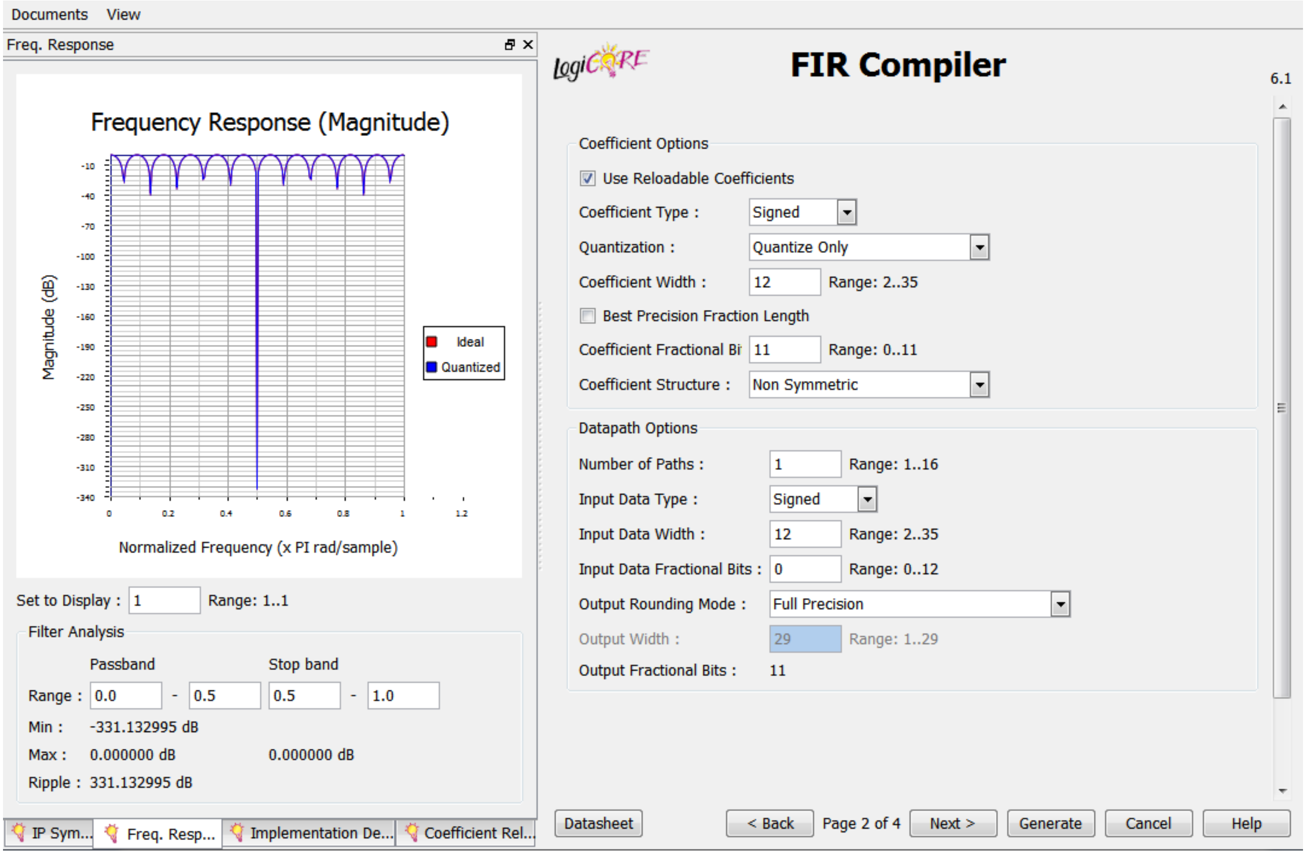
Select format : Sample Period

Input Sample Period : 1 Range: 1..10000000 Clock cycles

Input Sampling Frequency : 0.001 Range: 0.000001..300.0 MHz

Clock Frequency : 250.0 Range: 0.001..300.0 MHz

b)



Set to Display : 1 Range: 1..1

Filter Analysis

	Passband		Stop band	
Range :	0.0	- 0.5	0.5	- 1.0
Min :	-331.132995 dB			
Max :	0.000000 dB		0.000000 dB	
Ripple :	331.132995 dB			

Coefficient Options

☑ Use Reloadable Coefficients

Coefficient Type : Signed

Quantization : Quantize Only

Coefficient Width : 12 Range: 2..35

☐ Best Precision Fraction Length

Coefficient Fractional Bi 11 Range: 0..11

Coefficient Structure : Non Symmetric

Datapath Options

Number of Paths : 1 Range: 1..16

Input Data Type : Signed

Input Data Width : 12 Range: 2..35

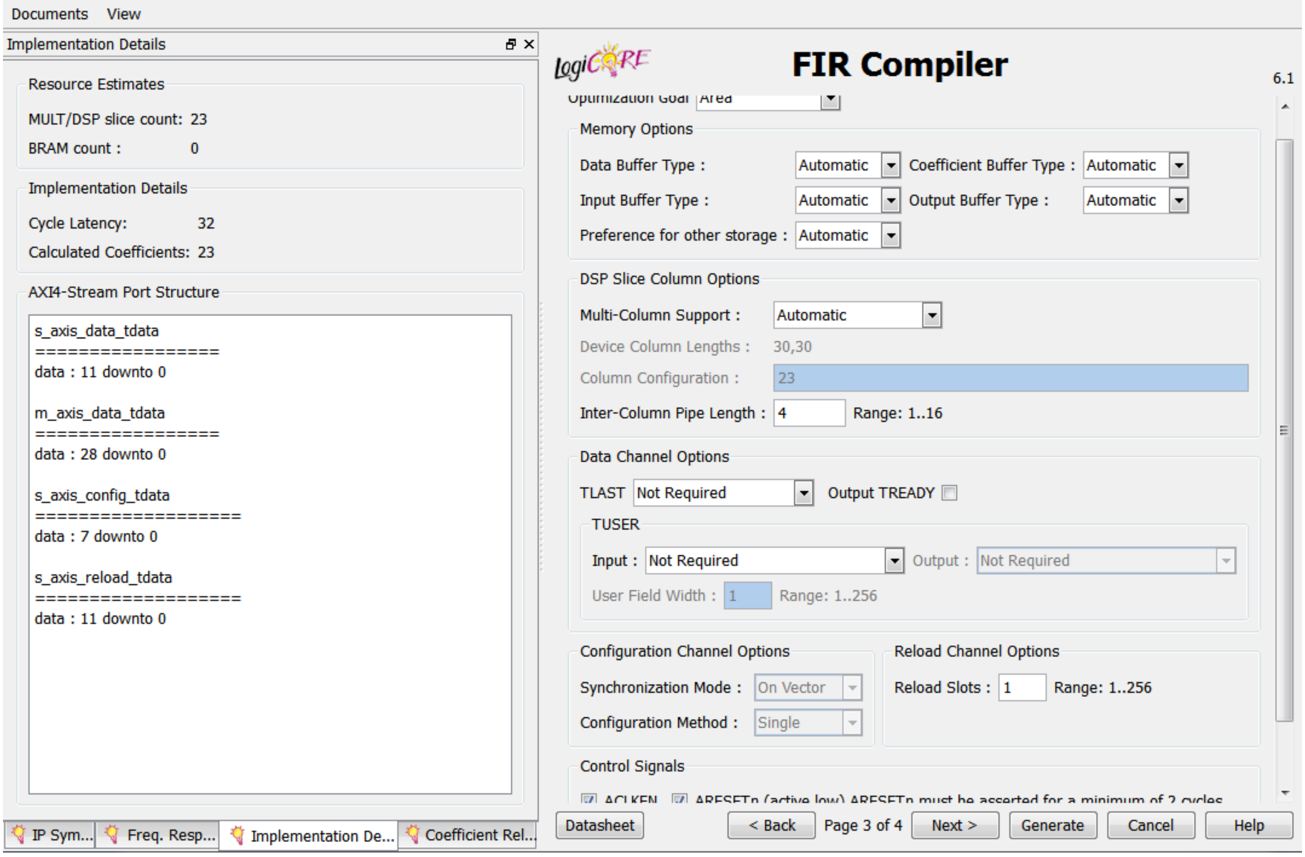
Input Data Fractional Bits : 0 Range: 0..12

Output Rounding Mode : Full Precision

Output Width : 29 Range: 1..29

Output Fractional Bits : 11

c)



Resource Estimates

MULT/DSP slice count: 23

BRAM count : 0

Implementation Details

Cycle Latency: 32

Calculated Coefficients: 23

AXI4-Stream Port Structure

```
s_axis_data_tdata
=================
data : 11 downto 0

m_axis_data_tdata
=================
data : 28 downto 0

s_axis_config_tdata
===================
data : 7 downto 0

s_axis_reload_tdata
===================
data : 11 downto 0
```

Optimization Goal Area

Memory Options

Data Buffer Type : Automatic Coefficient Buffer Type : Automatic

Input Buffer Type : Automatic Output Buffer Type : Automatic

Preference for other storage : Automatic

DSP Slice Column Options

Multi-Column Support : Automatic

Device Column Lengths : 30,30

Column Configuration : 23

Inter-Column Pipe Length : 4 Range: 1..16

Data Channel Options

TLAST Not Required Output TREADY ☐

TUSER

Input : Not Required Output : Not Required

User Field Width : 1 Range: 1..256

Configuration Channel Options

Synchronization Mode : On Vector

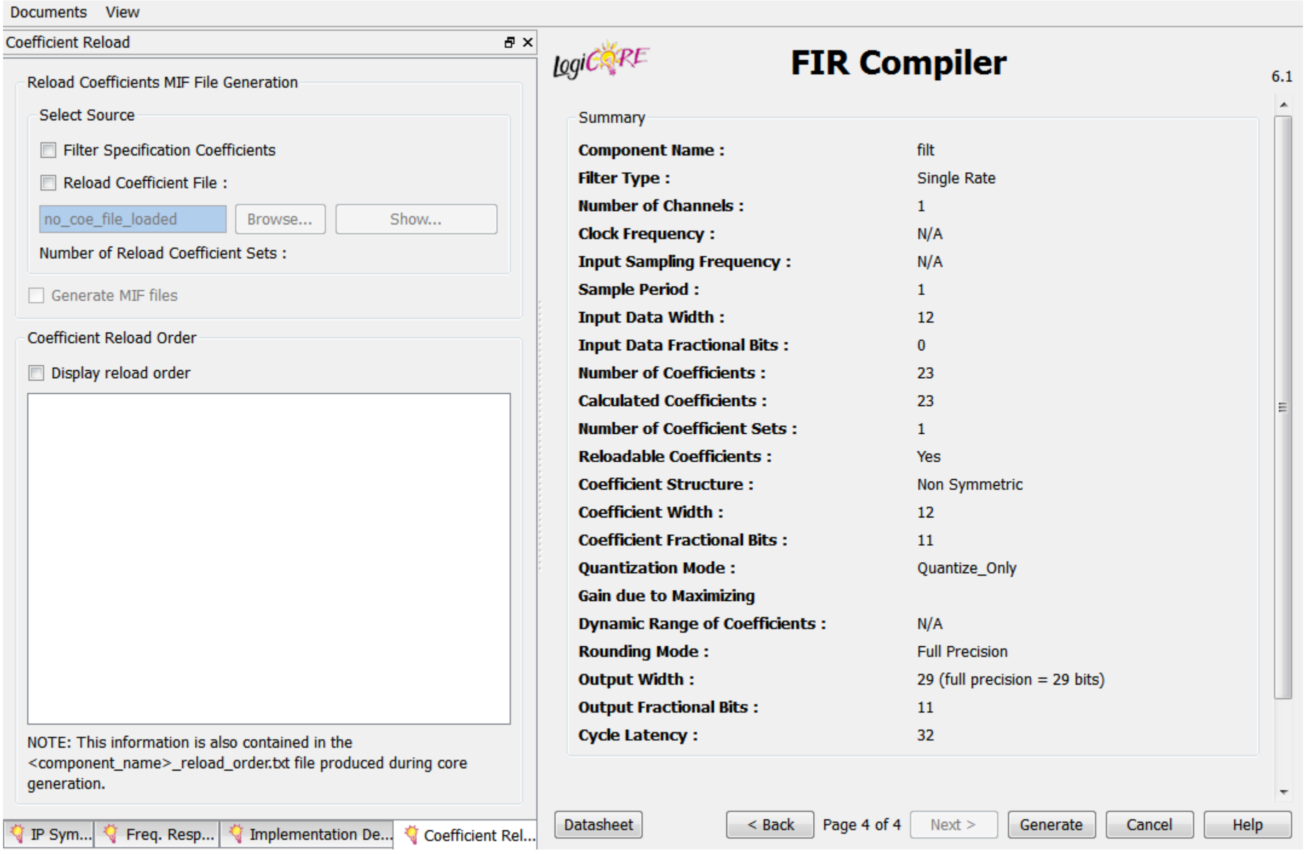
Configuration Method : Single

Reload Channel Options

Reload Slots : 1 Range: 1..256

Control Signals

☑ ACLKEN ☑ ARESETn (active low) ARESETn must be asserted for a minimum of 2 cycles d)



Reload Coefficients MIF File Generation

Select Source

□ Filter Specification Coefficients

□ Reload Coefficient File :

no_coe_file_loaded Browse... Show...

Number of Reload Coefficient Sets :

□ Generate MIF files

Coefficient Reload Order

□ Display reload order

NOTE: This information is also contained in the <component_name>_reload_order.txt file produced during core generation.

Summary

Component Name :	filt
Filter Type :	Single Rate
Number of Channels :	1
Clock Frequency :	N/A
Input Sampling Frequency :	N/A
Sample Period :	1
Input Data Width :	12
Input Data Fractional Bits :	0
Number of Coefficients :	23
Calculated Coefficients :	23
Number of Coefficient Sets :	1
Reloadable Coefficients :	Yes
Coefficient Structure :	Non Symmetric
Coefficient Width :	12
Coefficient Fractional Bits :	11
Quantization Mode :	Quantize_Only
Gain due to Maximizing	
Dynamic Range of Coefficients :	N/A
Rounding Mode :	Full Precision
Output Width :	29 (full precision = 29 bits)
Output Fractional Bits :	11
Cycle Latency :	32

Figure B.3 – Fir Filter IP core parameters configuration. Pages a) 1, b) 2, c) 3 and d) 4.

Appendix C – System Layout

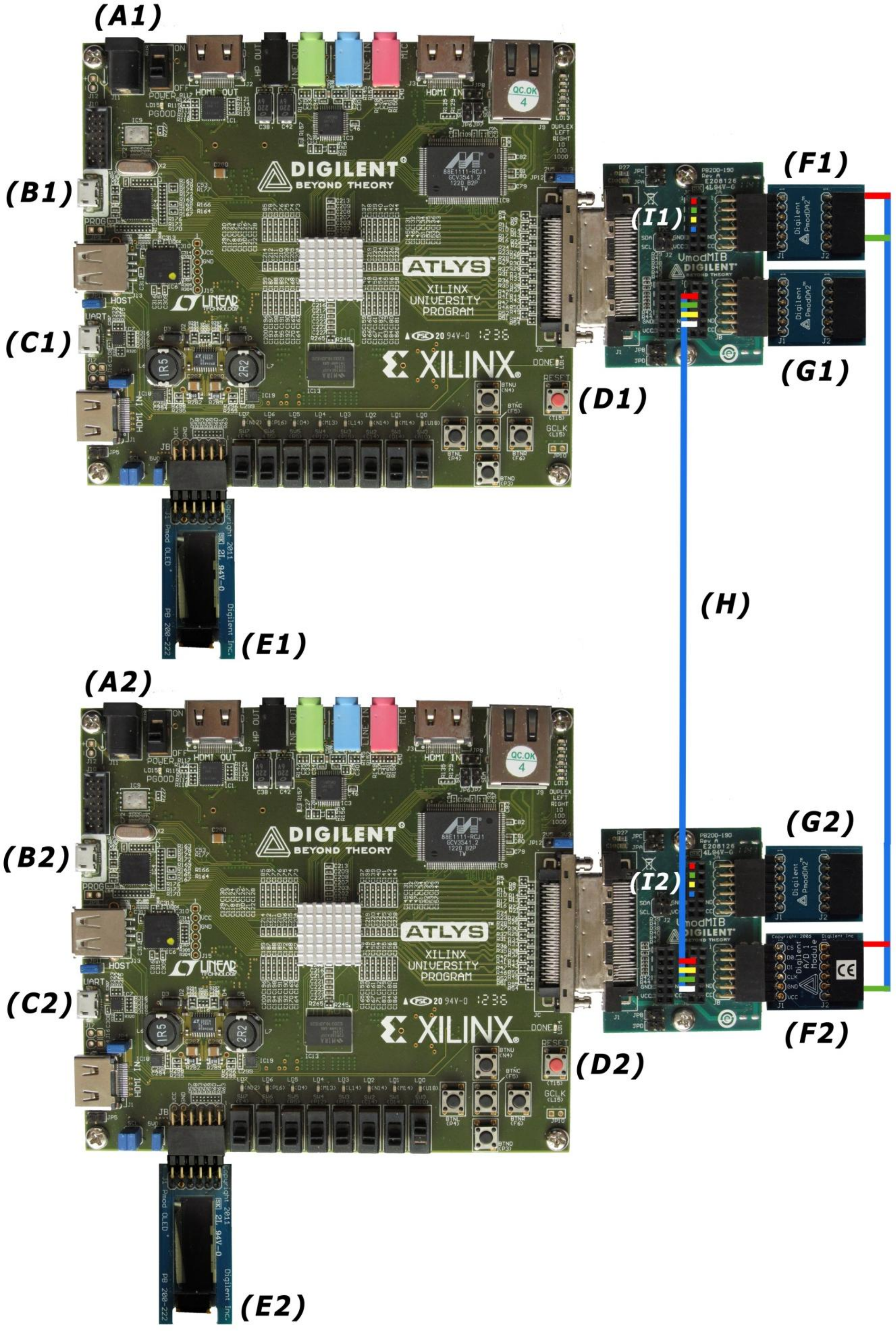


Figure C.1 – System Layout

The figure C.1 depicts the connections and the peripherals that must be present before running the system. In table C.1 is presented the meaning of the letters shown in figure C.1, where 1 and 2 correspond to master and slave FPGA, respectively. Is important to mention that the symbol rate (in the (I) point) has a duty-cycle equal to $\frac{1}{2\times Sampling\ \ Factor}$, where the sampling factor is defined by the user.

Letter	Meaning
A	Power Input
B	Program Input
C	UART input
D	Reset button
E	OLed Display
F	Chain DAC (F1) and ADC (F2); Green wire: ground; Red wire: channel 1
G	DAC Probes
H	Red wire: Master FIFO Write Enable Yellow and Green Wires: serial communication White wire: Ground
I	Red output (VmodMIB - C1): Symbol rate Green output (VmodMIB – C2): Data Generator LSB Yellow output (VmodMIB – C3): Generator Seed Detection Blue output (VmodMIB – C4): Sampling Frequency

Table C.1.1 – Layout meanings

References

[1] Ian A. Glover and Peter M. Grant, *Digital Communications*, 3rd ed. Harlow, England/Essex: Pearson Education Limited, 2010, pp. 55-64; 214-220; 255-259; 238-249.

[2] Saul A. Teukolky, William T. Vetterling and Brlan P. Flannery William H. Press, *Numerical Recipes The Art of Scientific Computing*, 3rd ed. USA/New York: Cambridge University Press, 2007, pp. 380–386; 362-363.

[3] B. P. Lathi, *Modern Digital and Analog Communication Systems*, 3rd ed. USA/New York: Oxford University Press, 1998, pp. 310-316.

[4] Ronald W. Schafer and Mark A. Yoder James H. McClellan, *Signal Processing First*, International ed.: Pearson Education, 2003, pp. 101-125.

[5] Q. Liu and S.W. Ellingson, "Effect and Correction of Unequal Cable Losses and Dispersive Delays on Delay-and-Sum Beamforming," April 25 2012.

[6] John F. Wakerly, *Digital Design Principals and Practices*, 4th ed.: Pearson Prentice Hall, 2005, pp. 11-15.

[7] [Online]. http://www.xilinx.com/support/documentation/user_guides/ug384.pdf

[8] [Online]. http://www.xilinx.com/support/documentation/sw_manuals/edk10_est_rm.pdf

[9] [Online]. http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/mb_ref_guide.pdf

[10] Digilent, Digilent PmodDA2™ Digital To Analog Module Converter Board Reference Manual, September 25, 2006.

[11] Digilent Romania, PmodDA2™ Reference Component, December 3, 2008.

[12] Digilent, Digilent PmodAD1™ Analog To Digital Module Converter Board Reference Manual, December 6, 2011.

[13] Xilinx, LogiCORE IP FIFO Generator v7.2, September 21, 2010, Product Specification.

[14] Xilinx, LogiCORE IP FIR Compiler v6.1, December 14, 2010, Product Specification.

[15] Paul B. Crilly, Janet C. Rutledge A. Bruce Carlson, *Communications Systems An Introduction to Signals and Noise in Electrical Communication*, 4th ed.: McGraw-Hill, 2002, pp. 437-447.

[16] Micro-Coax. [Online]. http://www.rf-microwave.com/datasheets/2916_MICRO-COAX_UT-141-75_01.pdf

[17] Steve Winder, *Analog and Digital Filter Design*, 2nd ed.: Newnes, 2002, pag. 423.